НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

“Інститут прикладного системного аналізу”

(повна назва інституту/факультету)

Системного проектування

(повна назва кафедри)

«На правах рукопису»

УДК 004:004.457

«До захисту допущено»

Завідувач кафедри

_____ А.І.Петренко

(підпис) (ініціали, прізвище)

“___”_____2018 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 122 – комп’ютерні науки та інформаційні технлогії

(код і назва спеціальності)

на тему: Побудова рекомендаційної системи для медіа-платформ з використанням методів глибокого навчання

Виконав (-ла): студент (-ка) __6__ курсу, групи _ДА-62м_

(шифр групи)

Мазурік Олексій Юрійович _____

(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к. т. н., доцент Кисельов Г.Д. _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розробка стартап-проекту к. т. н., доцент Кисельов Г.Д. _____

(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____ _____

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2018 року

Національний технічний університет України

«Київський політехнічний інститут

імені Ігоря Сікорського»

Інститут/факультет ННК "Інститут прикладного системного аналізу"

Кафедра Системного проектування

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 122 – комп'ютерні науки та інформаційні технології (Системне проектування сервісів)

(код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ А.І.Петренко

(підпис) (ініціали, прізвище)

«___»_____2018 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Мазуріку Олексію Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема дисертації Побудова рекомендаційної системи для медіа-платформ з використанням методів глибокого навчання

науковий керівник дисертації _Кисельов Генадій Дмітрович, к.т.н., доц.

затверджені наказом по університету від 27.03 2018 р. №1028-с

2. Строк подання студентом дисертації 10.05.2018

3. Об'єкт дослідження _ Багатошарові нейронні мережі, моделі Word2Vec, SkipGram та CBoW.

4. Предмет дослідження (Вихідні дані – для магістерської дисертації

за освітньо-професійною програмою) _методи машинного навання для побудови word embeddings, інструменти для побудови моделей PyTorch

5. Перелік завдань, які потрібно розробити _____

1. Дослідження існуючих методів глибокого навчання.
2. Дослідження уже існуючих рішень щодо обраного стандарту.
3. Вибір мови та розробка рекомендаційної системи.
4. Розробка стартап-проекту.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу _____ __
презентація на тему «Побудова рекомендаційної системи для медіа-платформ з використанням методів глибокого навчання»_____

7. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розробка стартап-проекту	Кисельов Г.Д., доц.		

9. Дата видачі завдання ___01.02.2018_____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.02.2018	
2	Огляд стану предметної області	15.02.2018	
3	Дослідження існуючих методів глибокого навчання	05.03.2018	
4	Аналіз існуючих рішень	30.03.2018	
5	Розробка векторних представлень відео	10.04.2018	
6	Реалізація запропонованого методу	17.04.2018	
7	Побудова рекомендаційної системи відео	30.04.2018	
8	Отримання допуску до захисту та подача роботи в ДЕК	10.05.2018	

Студент _____ ____О.Ю. Мазурік____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ ____Г.Д. Кисельов____
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

РЕФЕРАТ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

виконаної на тему: Побудова рекомендаційної системи

для медіа-платформ з використанням методів глибокого навчання

студентом: Мазуріком Олексієм Юрійовичем

Загальний обсяг роботи: 103 сторінки, 39 ілюстрацій, 17 таблиць, перелік посилань із 20 найменувань, 1 додаток на 12 сторінках.

Актуальність теми

Рекомендаційні системи з’явилися на сучасному ринку ІТ як механізм для заміни статичному списку рекомендацій при пошуку або покупках на веб-сайтах. Ці системи формують рейтинговий перелік об’єктів (товарів, фільмів, музичних композицій) на основі різних критеріїв: релевантність, популярність, історія оцінок тощо. Оскільки інформації у сучасному світі стало багато, тому важливо мати якісні інструменти для персоналізованої фільтрації великих даних, якими і виступають рекомендаційні системи.

З великою популяризацією та швидким розвитком машинного навчання штучний інтелект відповідає за ядро будь-якої серйозної персоналізованої системи. Тому зараз побудова рекомендаційної системи з використанням сучасних методів глибокого навчання – побудови моделей на основі багатошарових нейронних мереж – є нагальним питанням.

Мета та задачі дослідження

Метою даної роботи є пошук шляхів удосконалення існуючих рішень побудови рекомендаційних систем для медіа платформ із застосуванням нових підходів глибокого навчання. Задачею дослідження є реалізація системи рекомендацій медіа-контенту, яка використовує нові та більш ефективні методи, що мають переваги перед відомими методами без використання нейронних мереж.

Вирішення поставлених завдань та досягнуті результати

Було запропоновано використати модель SkipGram, що використовує багатошарові нейронні мережі, для пошуку семантичних зв'язків між різними відео. Це застосування є досить нестандартним, оскільки основна область застосування алгоритму Word2Vec є пошук семантичних зв'язків між словами. Таке використання моделі дозволяє побудувати точні векторні представлення відео у 32-вимірному просторі семантичних зв'язків. Також дана модель дозволяє легко побудувати рекомендаційну систему медіа-контенту на основі отриманих даних.

Дану модель було випробувано на наборі даних, що містить сесії переглядів користувачами відео синтетичного відео-сервісу. Також було проведено експеримент з побудови рекомендаційної системи з використанням даних моделі.

Об'єкт дослідження

Багатошарові нейронні мережі, моделі Word2Vec, SkipGram та CBoW.

Предмет дослідження

Методи машинного навчання для побудови word embeddings.

Методи дослідження

Досліджуються та застосовуються дво- та тришарові нейронні мережі. Для побудови векторних представлень досліджуються моделі SkipGram та CBoW.

Розроблене рішення використовує сучасні моделі машинного навчання та підходи до розробки їх архітектур; методи та техніки навчання, покращення точності та стійкості; а також бібліотеки для їх реалізації та розподіленого тренування.

Наукова новизна

Було запропоновано архітектуру моделі SkipGram для пошуку семантичних зв'язків між різними відео. Дану модель Word2Vec ще не було використано в опублікованих роботах для побудови рекомендацій медіа-контенту, оскільки першочерговою задачею Word2Vec є пошук зв'язків між словами. Дана модель показала кращі результати, ніж статичні рекомендації на основі оцінок, оскільки за

допомогою неї можна побачити багато прихованих зв’язків між відео. Також було запропоновано методі для покращення результатів роботи моделі з використанням ще кращих методів для побудови моделі Word2Vec, мета-даних відео (назва, ключові слова, оцінки) та самого контенту відео (згорткові нейронні мережі дадуть змогу знайти залежність і тут, але в цьому випадку задачу буде зведено до вирішення нової вже Computer Vision задачі).

Практичне значення одержаних результатів

Розроблена модель дозволяє виконувати пошук прихованих зв’язків між різними відео для побудови точної та релевантної рекомендаційної системи, яку можна вбудувати до будь-якої існуючої веб-системи, оскільки реалізація має детально описане API та зручний користувацький інтерфейс (UI) на Redux.

Апробації результатів дисертації

Результати дослідження оприлюднені в №9 Мультидисциплінарного Міжднародного наукового журналу «Інтернаука» за 2018 та 2015 роки.

Публікації

Мазурік О.Ю. Побудова рекомендаційної системи відео за допомогою моделей класу Word2Vec / Мазурік Олексій Юрійович // Міжнародний науковий журнал «Інтернаука». – 2018 . – №9 . – С. 87-97.

Мазурік О.Ю. Покращення результатів роботи рекомендаційних алгоритмів за допомогою алгоритму SVD / Мазурік Олексій Юрійович // Міжнародний науковий журнал «Інтернаука». – 2015 . – №9 . – С. 41-51.

Ключові слова

Машинне навчання, нейронні мережі, рекомендації відео, модель Word2Vec, згорткові нейронні мережі, модель SkipGram, векторні представлення, PyTorch, TensorFlow, розподілене тренування.

РЕФЕРАТ МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

выполненной на тему: Построение рекомендательной системи

для медиа-платформ с использованием методов глубокого обучения

студентом: Мазуриком Алексеем Юрьевичем

Общий объем работы 103 страницы, 39 иллюстраций, 17 таблиц, список литературы из 20 наименований, 1 приложение на 12 страницах.

Актуальность темы

Рекомендательные системы появились на современном ИТ-рынке как механизм для замены статического списка рекомендаций при поиске или покупках на веб-сайтах. Эти системы формируют рейтинговый перечень объектов (товаров, фильмов, музыкальных композиций), основываясь на различных критериях: релевантность, популярность, история оценок и тому подобное. Поскольку информации в современном мире стало много, важно иметь качественные инструменты для персонализированной фильтрации больших данных, которыми и выступают рекомендательные системы.

При текущей популяризации и быстром развитии машинного обучения искусственный интеллект отвечает за ядро любой серьезной персонализированной системы. Поэтому сейчас построение рекомендательной системы с использованием современных методов глубокого обучения - построения моделей на основе многослойных нейронных сетей – очень актуальный вопрос.

Цель и задачи исследования

Целью данной работы является поиск путей усовершенствования существующих решений построения рекомендательных систем для медиа-платформ с применением новых подходов глубокого обучения. Задачей исследования является реализация системы рекомендаций медиа-контента, которая использует новые и более эффективные методы, которые имеют преимущества перед известными статичными методами без использования нейронных сетей.

Решение поставленных задач и достигнутые результатах

Было предложено использовать модель SkipGram, которая использует многослойные нейронные сети для поиска семантических связей между разными видео. Такое использование данной модели является достаточно нестандартным, поскольку основная область применения алгоритма Word2Vec – поиск семантических связей между словами. Такое «новое» использование модели позволяет построить точные векторные представления видео в 32-мерном пространстве семантических связей. Также данная модель позволяет легко построить рекомендательную систему медиа-контента на основе полученных данных.

Данная модель была опробована на наборе данных, содержащем сессии просмотров пользователями видео синтетического видео-сервиса. Также был проведен эксперимент по построению рекомендательной системы с использованием данных модели.

Объект исследования

Многослойные нейронные сети, модели Word2Vec, SkipGram и CBoW.

Предмет исследования

Методи машинного обучения для построения word embeddings.

Методи исследования

Исследуются и применяются двухслойные и трехслойные нейронные сети. Для построения векторных представлений исследуются модели SkipGram и CBoW.

Разработанное решение использует современные модели машинного обучения и подходы к разработке их архитектур; методы и техники обучения, улучшение точности и устойчивости; а также библиотеки для их реализации и распределенных тренировок.

Научная новизна

Было предложено использовать архитектуру модели SkipGram для поиска семантических связей между различными видео. Данная модель Word2Vec еще не

использовалась в опубликованных работах с открытым исходным кодом для построения рекомендаций медиа-контента, поскольку первоочередной задачей Word2Vec является поиск связей между словами. Также были предложены методы для улучшения результатов работы модели с использованием оптимизированных методов для построения модели Word2Vec, мета-данных видео (название, ключевые слова, оценки) и самого контента видео (сверточные нейронные сети позволят найти зависимость и здесь, но в этом случае задачу будет сведено к решению новой, уже Computer Vision задачи).

Практическое значение полученных результатов

Разработанная модель позволяет выполнять поиск скрытых связей между различными видео для построения точной и релевантной рекомендательной системы, которую можно встроить в любую существующую веб-систему, поскольку реализация имеет подробно описанное API и удобный пользовательский интерфейс (UI) на Redux.

Апробация результатов диссертации

Результаты исследования размещены в №9 Мультидисциплинарного Международного научного журнала «Интернаука» за 2018 та 2015 год.

Публикации

Мазурік О.Ю. Побудова рекомендаційної системи відео за допомогою моделей класу Word2Vec / Мазурік Олексій Юрійович // Міжнародний науковий журнал «Інтернаука». – 2018 . – №9 . – С. 87-97.

Мазурік О.Ю. Покращення результатів роботи рекомендаційних алгоритмів за допомогою алгоритму SVD / Мазурік Олексій Юрійович // Міжнародний науковий журнал «Інтернаука». – 2015 . – №9 . – С. 41-51.

Ключевые слова

Машинное обучение, нейронные сети, рекомендации видео, модель Word2Vec, сверточные нейронные сети, модель SkipGram, векторные представления, PyTorch, TensorFlow, распределенные тренировки.

ABSTRACT FOR MASTER'S THESIS

on: Building a recommender system

for media platforms using deep learning techniques

by: Mazurik Oleksii Yuriyovich

The thesis contains 103 pages, 39 figures, 17 tables, 20 references, and 1 appendix (12 pages).

Relevance

Recommender systems appeared on the modern IT-market as a tool for replacing the static list of recommendations for searching or shopping on the different web-platforms. These systems form a rated list of the objects (goods, films, songs), based on various criteria: relevance, popularity, history of assessments, and users' likes. As there is a lot of information in the modern world, it is important to have quality tools for personalized filtering of large data like recommender systems.

With the current popularization and rapid development of the machine learning, artificial intelligence is responsible for the core of any solid personalized system. Therefore, now the construction of a recommender system using modern techniques of in-depth training - building models based on multilayer neural networks - is a very important question to discuss.

Purpose

The purpose of this paper is to find the ways to improve existing solutions for building recommender systems for media platforms using new approaches to in-depth training. The goal of the study is to implement a system of media content recommendations that uses modern and effective methods that have advantages over known static methods without the use of neural networks.

Results

It was suggested to use the SkipGram model, which uses multi-layer neural networks to search for semantic links between different videos. This usage of this model is quite

unusual since the main scope of the Word2Vec algorithm is the search for semantic links between the words. This "innovative" use of the model makes it possible to build accurate vector representations of video in the 32-dimensional space of semantic connections. Also, this model makes it easy to build a recommendatory system of media content based on the received data.

This model was tested on the dataset containing sessions of the users' video-viewing on the synthetic video service. The experiment was also conducted to build the recommender system using the model data.

Object of research

Multilayer neural networks; Word2Vec, SkipGram and CBoW models.

Subject of research

Machine learning methods for building word embeddings.

Research methods

Two-layer and three-layer neural networks are explored and applied. For the construction of vector representations, the SkipGram and CBoW models are investigated.

The developed solution uses modern machine learning models and approaches to architecture their architecture development; methods and techniques for training, improving accuracy and robustness; libraries for implementation and distributed training.

Scientific novelty

It was suggested to use the SkipGram model architecture to search for the semantic links between the different videos. This model of Word2Vec has not yet been used in the published open source paper to build recommendations for media content since Word2Vec's primary task is to search for links between words. This model showed better results than static recommendations based on the estimates because with it you can see a lot of hidden links between the videos. Some methods to improve the model's performance using optimized techniques for building a Word2Vec model have also been proposed, meta-data video (title, keywords, estimates) and the video content itself (convolutional neural

networks will find the dependency here, but in this case the task will be reduced to solving the new combined NLP + Computer Vision task).

Practical value

The developed model allows you to search for hidden links between different videos to build an accurate and relevant recommender system that can be embedded into any existing web-system since the implementation has a detailed and simple API and user-friendly interface (UI) on Redux.

Approbation of research results

Research results presented at the Issue #9 of the International science journal “Internauka” in the 2018 and 2015.

Publications

Mazurik O. Building a recommender system using Word2Vec models / Mazurik Oleksii Yuriyovich // International science Journal «Internauka». – 2018 . – №9 . – С. 87-97.

Mazurik O. Improvement of recommendation systems with SVD algorithm / Mazurik Oleksii Yuriyovich // International science Journal «Internauka». – 2015 . – №9 . – С. 41-51.

Keywords

Machine learning, neural networks, video recommendations, Word2Vec model, convolutional neural networks, SkipGram model, word embeddings, PyTorch, TensorFlow, distributed training.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – application programming interface, прикладний програмний інтерфейс

ШНМ– Штучна нейронна мережа

WB – Word Embedding, Векторне представлення

NS – Negative Sampling

JS – JavaScript, мова програмування

BoW – Алгоритм Bag of Words

CBoW – Алгоритм Continuous Bag of Words

Word2Vec – Алгоритм переводу слова (word) до вектора (vec)

ВСТУП

Протягом значного проміжку часу в усьому світі швидкими темпами збільшується кількість інформації. Люди кожного дня сприймають та фільтрують вхідний потік інформації, що надходить з різних джерел: робота, побутові проблеми, популярні джерела інформації тощо. Після винайдення мережі Інтернет кількість такої інформації стала стрімко зростати, з’явилася велика кількість сервісів для надання користувачам всього необхідного для комфортного життя. [1]

За останню декаду набули значної популярності інтернет-сервіси, що пропонують товари всіх можливих видів (інтернет-магазини), інформацію на будь-який смак (інтернет-журнали, новини, книги, статті) тощо. Користувачу стало надзвичайно важко орієнтуватися в каталогах товарів та списках статей, навіть із вбудованим пошуком та фільтрацією, оскільки дуже важко зробити вибір при настільки великому об’ємі інформації.[1]

Рекомендаційні системи з’явилися на сучасному ринку IT як механізм для заміни статичному списку рекомендацій при пошуку або покупках на веб-сайтах. Ці системи формують рейтинговий перелік об’єктів (товарів, фільмів, музичних композицій) на основі різних критеріїв: релевантність, популярність, історія оцінок тощо. [1]

Такі системи почали широко використовувати існуючі інтернет-компанії в рамках інтернет-маркетингу. За допомогою прогнозування рекомендацій вони мають на меті збільшити конверсію користувачів до конкретного сервісу. Також, при розробці рекомендаційної системи з релевантними рекомендаціями, що заслужили довіру користувачів, можна розміщувати серед цих рекомендацій інші товари, що рекламуються.

Але, незважаючи на те, що рекомендаційні системи досить недавно з'явилися на ринку, вже існує безліч способів їх покращити. З розвитком досліджень у сфері Big Data та Machine Learning, все більше компаній починає використовувати нейронні мережі для покращення якості рекомендацій. Використовуються всі дані, які

користувач залишає на сайті – історія переглядів, кліків, оцінок тощо. Зараз набуває популярність алгоритм побудови моделі Word2Vec (алгоритм пошуку семантичних зв'язків між словами). Його починають використовувати для пошуку семантичних зв'язків між різними типами медіа-даних. Саме експеримент з дослідження можливості використання цього алгоритму для медіа-даних буде основним предметом дослідження даної роботи.

У *теоретичній частині* даної роботі буде досліджено загальні методи побудови нейронних мереж (теоретичні відомості, методи оптимізації), різні типи нейронних мереж (згорткові, рекурентні) та різні типи функцій активації (softmax). Також будуть досліджені алгоритми пошуку семантичних зв'язків між словами , такі як SkipGram та CBoW (способи реалізації моделі Word2Vec), а також способи їх оптимізації (алгоритми Negative Sampling та Hierarchical SoftMax).

Також для наочності будуть проаналізовані поточні рекомендаційні системи відомих інтернет-сервісів, таких як YouTube (Google) та Airbnb. Буде розглянуто та систематизовано структури рекомендаційних систем цих сервісів та дослідження спроб покращення цих систем (алгоритму Word2Vec) задля збільшення конверсії.

У *практичній частині* роботи будуть досліджені основні інструменти для реалізації алгоритмів глибокого навчання, такі як PyTorch та TensorFlow. Буде проаналізована доречність їх використання у різних ситуаціях. На базі наведених вище теоретичних відомостей буде проведена серія експериментів для побудови векторних представлень відео. Дані векторні представлення (video embeddings) будуть використані для побудови рекомендаційної системи медіа, яку можна буде легко вбудувати до діючого веб-сайту.

Завданням дипломного проекту є дослідження основних алгоритмів глибокого навчання, побудова рекомендаційної системи медіа з використанням цих алгоритмів (побудова згорткової нейронної мережі для моделі Word2Vec – Videoid2Vec) та оцінка роботи даної системи.

1 МЕТОДОЛОГІЯ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Рекомендаційна система – це система, що рекомендує товари користувачам серед величезного потоку інформації, в залежності від їх потреб. Товари – елементи конкретного сервісу, до яких користувач має інтерес: фільми, ресторани, книги, статті і т.ін. Інтереси користувачів можуть бути представлені декількома способами: із застосуванням оцінок, які користувачі надають товарам або за допомогою ключових слів кожного товару.[1]

Але зі зміною ринку та технологій з’являється все більше і більше інструментів для відстежування інтересів користувачів. Якщо раніше це були лише ключові слова товару та оцінки користувачів, то тепер можна відстежувати навіть сесії користувачів: де користувач гортає сторінку в конкретний момент часу, куди натискає. Вся ця інформація буде записана до анонімних логів, які можна буде використовувати для навчання нейронних мереж, щоб рекомендувати нові товари точніше.

Звичайно, щоб розібратися з усіма явними та неявними зв’язками, які можуть знаходитися у такій інформації, необхідно використовувати сучасні напрацювання в сфері машинного навчання – нейронні мережі.

1.1 Штучна нейронна мережа

Штучні нейронні мережі (ШНМ) - це програмна імплементація нейронних структур нашого мозку. Не будемо обговорювати складну біологію нашої голови, досить знати, що мозок містить нейрони, які є свого роду органічними перемикачами. Вони можуть змінювати тип переданих сигналів в залежності від електричних або хімічних сигналів, які в них передаються. Нейронна мережа у людському мозку - величезна взаємопов'язана система нейронів, де сигнал, який передається одним нейроном, може передаватися у тисячі інших нейронів. Навчання відбувається через повторну активацію деяких нейронних з'єднань. Через це збільшується імовірність виведення потрібного результату при відповідній вхідній інформації (сигналах).

Такий вид навчання використовує зворотний зв'язок - при правильному результаті нейронні зв'язки, які виводять його, стають більш щільними. [4]

Штучні нейронні мережі імітують поведінку мозку у простішому вигляді. Вони можуть бути навчені контрольованим та неконтрольованим шляхами. У контрольованій ШНМ, мережа навчається шляхом передавання відповідної вхідної інформації та прикладів вихідної інформації. Наприклад, спам-фільтр у електронній поштовій скриньці: вхідною інформацією може бути список слів, які зазвичай містяться у спам-повідомленнях, а вихідною інформацією - класифікація для відповідного повідомлення (спам, чи не спам). Такий вид навчання додає ваги зв'язкам ШНМ, але це буде обговорено пізніше. [2]

Неконтрольоване навчання у ШНМ намагається "змусити" ШНМ "зрозуміти" структуру переданої вхідної інформації "самостійно". [3]

1.1.1 Структура Штучної нейронної мережі

1.1.1.1 Поняття штучного нейрона

Біологічний нейрон імітується у ШНМ через активаційну функцію. У задачах класифікації (наприклад визначення спам-повідомлень) активаційна функція повинна мати характеристику "вмикача". Іншими словами, якщо вхід більше, ніж деяке значення, то вихід повинен змінювати стан, наприклад з 0 на 1 або -1 на 1. Це імітує "включення" біологічного нейрону. У якості активаційної функції зазвичай використовують сигмоїдну функцію [3]:

$$f(z) = \frac{1}{1 + \exp(-z)} \tag{1}$$

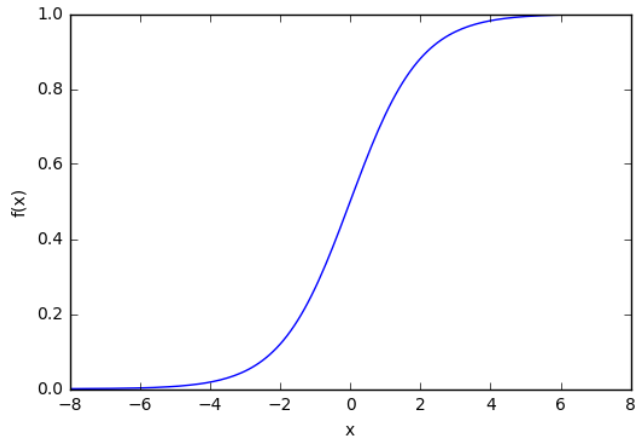

Рис. 1.1 – Графік функції сигмоїду

З графіку можна побачити, що функція «активаційна» – вона росте з 0 до 1 з кожним збільшенням значення ***x***. Сигмоїдна функція є гладкою і неперервною. Це означає, що функція має похідну, що у свою чергу є дуже важливим фактором для навчання алгоритму. [3]

1.1.1.2 Поняття вузлів перцептрону

Як було згадано раніше, біологічні нейрони ієрархічно з'єднані в мережамі, де вихід одних нейронів є входом для інших нейронів. Ми можемо представити такі мережі у вигляді з'єднаних шарів з вузлами. Кожен вузол приймає зважений вхід, активує активаційну функцію для суми входів, та генерує вихід [2].

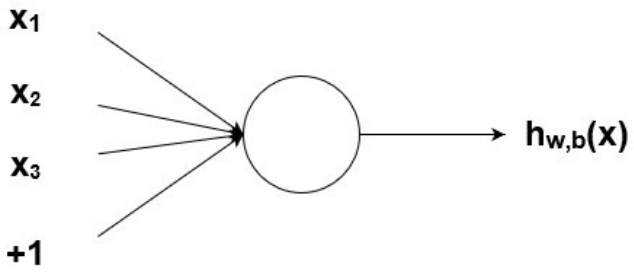

Рис. 1.2 – Простий вузол з кількома входами

Коло на рисунку 1.2 зображує вузол. Вузол є "місцерозташуванням" активаційної функції, він приймає зважені входи, сумує їх, а потім вводить їх в активаційну функцію. Вивід активаційної функції представлений через ***h***. Примітка: у деякій літературі *вузол* також називають *перцептроном [3]*.

Що є «вагою»? За вагу беруться числа (не бінарні), які потім множаться на вході і сумуються у вузлі. Іншими словами, зважений вхід у вузол має вигляд: [2]

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b, \tag{2}$$

де w_i - числові значення ваги (b ми будемо обговорювати пізніше). Ваги нам потрібні, вони є значеннями, які будуть змінюватись протягом процесу навчання. b є вагою елемента зміщення на +1, включення ваги b робить вузол гнучкішим. [4]

1.1.1.3 Зміщення байєсу у вузлів перцептрона

Роглянемо простий вузол, у якому є по одному входу та виходу:

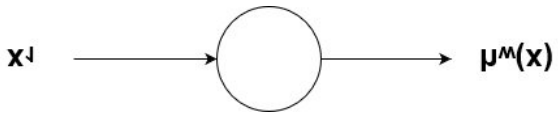

Рис. 1.3 – Простий вузол

Виходом для активаційної функції у цьому вузлі є просто $x_1 w_1$. На що впливає зміна в w_1 у цій простій мережі? [2]

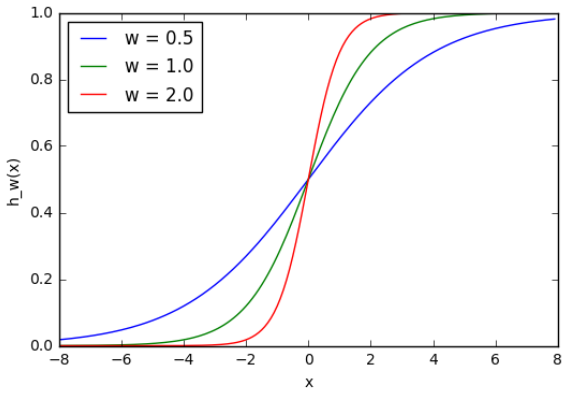

Рис. 1.4 – Графік ефекту збільшення ваги на простому вузлі

Тут можна побачити, що при зміні ваги змінюється також рівень нахилу графіка активаційної функції. Це корисно при моделюванні різних щільностей взаємозв'язків між входами та виходами. Але що робити, якщо ми хочемо, щоб вихід змінювався тільки при x більше 1? Для цього нам потрібне зміщення. Розглянемо таку мережу із зміщенням на вході: [3]

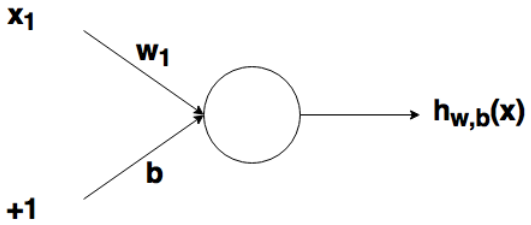

Рис. 1.5 – Складний вузол

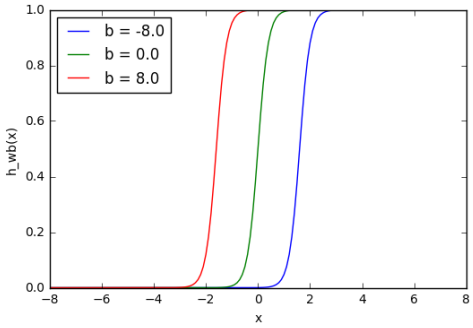

Рис. 1.6 – Ефект зміщення на складному вузлі

З графіку можна побачити, що змінюючи «вагу» зміщення ***b***, можна змінювати час запуску вузла. Зміщення дуже важливе у випадках, коли потрібно імітувати умовні відносини. [4]

1.1.2 Складена структура Штучної Нейронної Мережі

Вище було пояснено, як працює відповідний вузол/нейрон/персептрон. Але, як відомо, у повній нейронній мережі знаходиться багато таких взаємозв'язаних між собою вузлів. Структури таких мереж можуть приймати міріади різних форм, але найпоширеніша складається з вхідного шару, прихованого шару та вихідного шару. Приклад такої структури приведено нижче: [5]

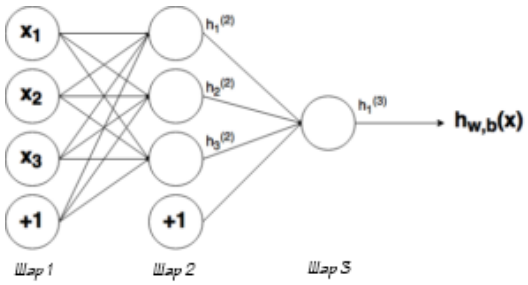

Рис. 1.7 – Три шари нейронної мережі

Ну рисунку 1.7 можна побачити три шари мережі - Шар 1 є вхідним шаром, де мережа приймає зовнішні вхідні дані. Шар 2 називають прихованим шаром - цей шар не є частиною ні входу, ні виходу. Примітка: нейронні мережі можуть мати декілька прихованих шарів, у даному прикладі було включено лише один шар для простоти. І нарешті, Шар 3 є вихідним шаром. Ви можете помітити, що між Шаром 1(Ш1) та Шаром 2(Ш2) існує багато зв'язків. Кожен вузол у Ш1 має зв'язок зі всіма вузлами у Ш2, при цьому від кожного вузла у Ш2 йде по одному зв'язку до єдиного вихідного вузла у Ш3. Кожен з цих зв'язків повинен мати відповідну вагу. [5]

1.1.3 Згорткова нейронна мережа

Згорткова нейронна мережа (ЗНМ) – тип багатошарової нейронної мережі, яка свою назву «згорткова мережа» отримала за назвою операції – згортка, вона часто використовується для обробки зображень і може бути описана наступною формулою: [5]

$$(f \times g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l] \qquad (3)$$

де **f** – вихідна матриця зображення; **g** – ядро (матриця) згортки.

Неформально цю операцію можна описати наступним чином – вікном розміру ядра **g** проходимо із заданим кроком (зазвичай 1) все зображення **f**: на кожному кроці поелементно множимо вміст вікна на ядро **g**, результат сумується і записується в таблицю результату. [6]

Ідея згорткових нейронних мереж полягає в чергуванні згорткових шарів (англ. convolution layers) і субдискретизуючих шарів (англ. subsampling layers, верств підвибірки). Структура мережі – односпрямована (без зворотних зв’язків), багатошарова (рис. 1.8). [5]

Модель згорткової мережі складається з трьох типів шарів: згорткові (convolutional) шари, субдискретизуючі (subsampling, підвибірка) шари і прошарки «звичайної» нейронної мережі – перцептрона. Архітектура згорткових нейронних

мереж реалізує три ідеї, які забезпечують інваріантність мережі до невеликих зрушень, змін масштабу і спотворень:[3]

- кожен нейрон отримує вхідний сигнал від локального рецептивного поля (local receptive fields) у попередньому шарі, що забезпечує локальну двовимірну зв'язність нейронів; [2]
- кожен прихований шар мережі складається з безлічі карт ознак, на яких всі нейрони мають загальні ваги (shared weights), що забезпечує інваріантність до зміщення і скорочення загальної кількості вагових коефіцієнтів мережі; [2]
- за кожним шаром згортки слідує обчислювальний шар, який здійснює локальне усереднення та підвибірку, що забезпечує зменшення розширення для карт ознак. [3]

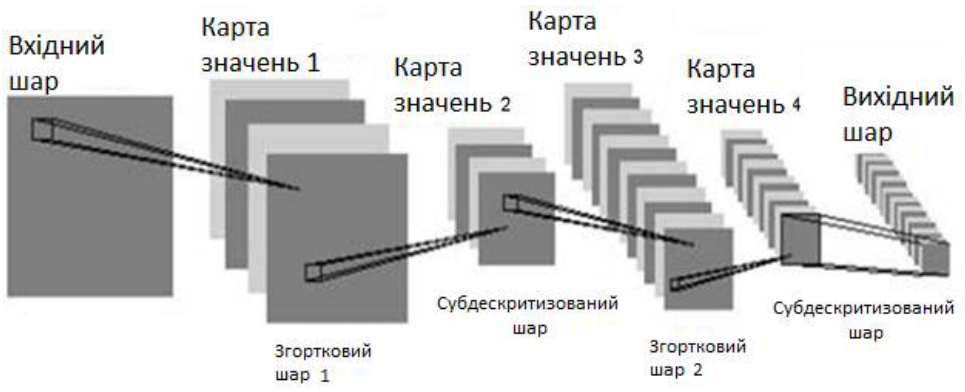

Рис. 1.8 – Структура згорткової мережі

Робота згорткової нейронної мережі забезпечується двома основними елементами. [5]

1) Фільтри (filters) (визначники ознак).

2) Карти ознак (feature maps).

Фільтр – це невелика матриця, що представляє ознаку, яку необхідно знайти на вихідному зображенні. За допомогою верхнього фільтра визначаються частини вихідного зображення з вертикальними лініями, нижній фільтр служить для визначення частин зображення з горизонтальними лініями.[5]

Безпосередньо процес визначення заснований на операції згортки фільтром оригінального зображення. Результати згортки, які визначають місце розташування ознак вихідного зображення, називаються картами ознак.[6]

Мета процесу згортки – зменшити розмірність карти ознак до такої міри, щоб з повним набором ознак могла працювати мережа прямого поширення (в більшості випадків багатошаровий персептрон). [5]

Згортковий шар реалізує ідею локальних рецептивних полів, тобто кожен вихідний нейрон з'єднаний тільки з певною (невеликою) областю вхідної матриці і таким чином моделює деякі особливості людського зору. Недоліками згорткових нейронних мереж (ЗНМ) є [3]:

- висока складність архітектури;
- повнозв'язаність;
- фіксована площа вікна шару згортки.

З метою підвищення ефективності роботи ЗНМ необхідно знайти оптимальні значення наступних параметрів [2]:

- кількість карт ознак;
- щільність зв'язків між картами ознак;
- розмір вікна;
- площа перекриття;
- початкова ініціалізація ваг.

1.2 Модель Word2Vec для побудови Word Emdedings

Для побудови рекомендаційної системи на основі згорткових нейронних мереж необхідно вміти обробляти великі масиви даних – тексти, мультимедіа тощо. Нещодавно була запропонована ідея застосування моделі Word2Vec для роботи з мультимедійними об'єктами, що є досить незвичайно, оскільки основна мета застосування даної моделі – пошук семантичних зв'язків між словами у великих корпусах документів. Перші приклади застосування будуть представлені у розділі 2,

де будуть розглянуті такі моделі для роботи з мультимедійними даними. Нижче буде наведені теоретичні відомості про модель Word2Vec [6].

1.2.1 Поняття Word Emdebbings

Основним інструментом сучасної обробки текстів наразі є розподілені представлення слів (distributed word representations, вони ж word embeddings). В цих представленнях кожному слову ставиться у відповідність вектор із дійсних чисел, елемент евклідового простору $\boldsymbol{R^d}$ для якогось $\boldsymbol{d}$ (зазвичай кілька сотень). Ці вектори далі служать входами для наступних моделей, а базове припущення полягає в тому, що геометричні відношення в просторі $\boldsymbol{R^d}$ будуть відповідати семантичним відношенням між словами, наприклад, найближчі сусіди слова в цьому просторі виявляються його синонімами або іншими тісно пов’язаними словами і т.д. [7]

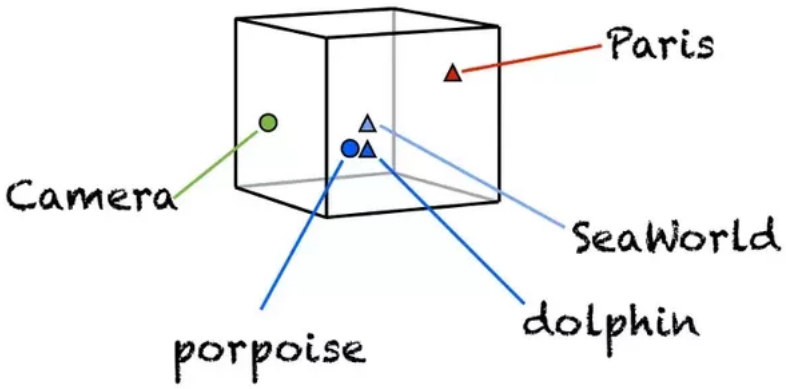

Рис. 1.9 - Приклад простору сусідніх слів

Отже, у нас є слова і є комп’ютер, який повинен з цими словами якось працювати. Питання - як комп'ютер буде працювати зі словами? Адже комп'ютер не вміє читати, і взагалі влаштований іншим чином, ніж людина. Найперша ідея, що приходить в голову - просто закодувати слова цифрами по порядку проходження в словнику. Ідея є продуктивною в своїй простоті - натуральний ряд нескінченний і можна перенумерувати всі слова, не побоюючись проблем. [7]

1.2.2 Типи зв’язків між словами

У ідеї закодувати слова цифрами по порядку є істотний недолік: слова в словнику знаходяться в алфавітному порядку, і при додаванні слова необхідно

перенумеровувати знову більшу частину слів. Але навіть це не є дуже складним. Дуже важливим є те, що написання слова за допомогою літер жодним чином не пов'язане з його змістом (цю гіпотезу ще в кінці XIX століття висловив відомий лінгвіст Фердинанд де Соссюр). Справді слова «півень», «курка» і «курча» мають дуже мало спільного між собою і стоять в словнику далеко один від одного, хоча очевидно мають значення самця, самки і дитинча одного виду птахів. Тобто ми можемо виділити два види схожості слів: лексичний і семантичний. Як можна побачити на прикладі з куркою, ці два типи подібностей не обов'язково перетинаються. Можна для наочності також привести зворотній приклад лексично близьких, але семантично далеких слів – «зола» та «золото». Щоб отримати можливість представити семантичну близькість, було запропоновано використовувати ***embedding*** (векторне представлення), тобто протиставити слову якийсь вектор, що відображає його значення в «просторі значень». [6]

1.2.3 Алгоритми One-hot-encoding та Bag of Words

Який найпростіший спосіб отримати вектор зі слова? Здається, що природньо буде взяти вектор довжини нашого словника і поставити одиницю тільки на тій позиції, що відповідає номеру слова в словнику. Цей підхід називається *one-hot encoding* (***OHE***). ***OHE*** все ще не має властивості семантичної близькості як можна побачити на рисунку знизу. Отже, необхідно знайти інший спосіб перетворення слів у вектори, але ***OHE*** алгоритм ще знадобиться. [7]

Рис. 1.10. Приклад one-hot-encoding

Значення одного слова може бути і не так важливо, тому що мова (і усна, і письмова) складається з наборів слів, які називаються текстами. Так що у випадку, коли захочеться якось змоделювати тексти, то необхідно взяти OHE-вектор кожного слова в тексті і скласти разом. Тобто на виході отримаємо просто кількість різних слів у тексті в одному векторі. Такий підхід називається «сумка слів» (*bag of words, BoW*),

оскільки втрачається всю інформацію про взаємне розташування слів у тексті (див. рис. 7). [7]

Незважаючи на втрату цієї інформації, таким способом тексти вже можна порівнювати. Наприклад за допомогою косинусної міри (коефіцієнт міри подібності). Косинусна міра подібності – це міра схожості між двома векторами предгільбертового простору (такого, на якому визначений скалярний добуток між двома векторами), яка використовується для підрахунку косинуса кута між ними. Якщо дано два вектори ознак A і B, то косинусна подібність ($\cos\theta$) може бути підрахована за допомогою скалярного добутку та норми векторів [6]:

$$similatiry = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \quad (4)$$

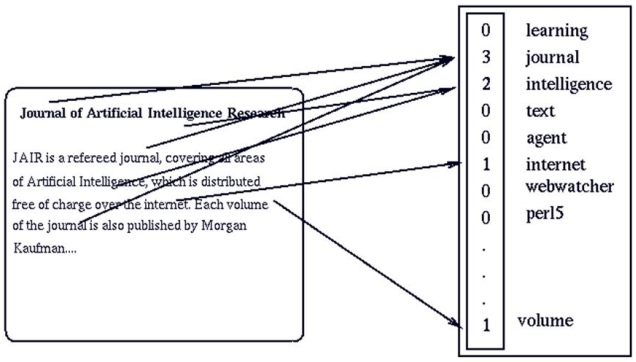

Рис. 1.11. Приклад використання bag-of-words

Можна піти далі і представити наш корпус (набір текстів) у вигляді матриці «слово-документ». Варто зазначити, що в області інформаційного пошуку (information retreival) ця матриця має назву зворотного індексу у тому сенсі, що прямий індекс виглядає як «документ-слово» є дуже незручним для швидкого пошуку. Але це виходить за рамки даного дослідження. [7]

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
computer	1	1	0	0	0	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
time	0	1	0	0	1	0	0	0	0
user	0	1	1	0	1	0	0	0	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0

Рис. 1.12. Приклад матриці слово-документ

Ця матриця приводить до тематичних моделей, де матрицю «слово-документ» намагаються представити у вигляді добутку двох матриць «слово-тема» та «тема-документ». У найпростішому випадку можна взяти цю матрицю і за допомогою SVD-розкладу отримати представлення слів через теми та документів через теми [6]:

$$\begin{matrix} & \boldsymbol{X} & & \boldsymbol{U} & & \boldsymbol{\Sigma} & & \boldsymbol{V^T} \\ & (\mathbf{d}_j) & & & & & & (\hat{\mathbf{d}}_j) \\ & \downarrow & & & & & & \downarrow \\ (\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} & = (\hat{\mathbf{t}}_i^T) \rightarrow & \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} \cdots \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} & \cdot \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_l \end{bmatrix} & \cdot \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_l &] \end{bmatrix} \end{matrix} , \tag{5}$$

де $\boldsymbol{t_i}$ – слова, $\boldsymbol{d_i}$ – документи.

1.2.4 Алгоритм TF-IDF

Нехай, маємо наступний корпус документів:

```
s = [“Mars has an athmosphere”, “Saturn’s moon Titan has its own athmosphere”, “Mars has two moons”, “Saturn has many moons”, “Io has cryo-vulcanoes”]
```

За допомогою SVD-розкладу виділимо лише перших два компоненти та побудуємо графік, що знаходиться на рис. 9. [7]

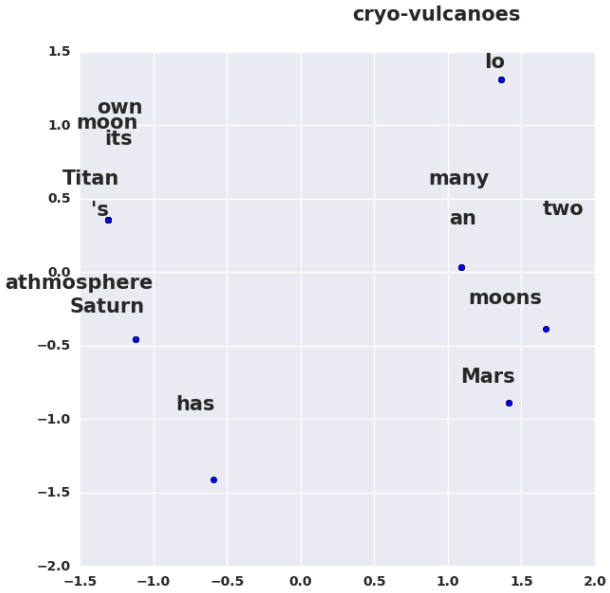

Рис. 1.13. Векторне представлення корпусу документів

Що цікавого на цьому рисунку? Те, що Титан і Іо - далеко один від одного, хоча вони обидва є супутниками Сатурна, але в нашому корпусі про це нічого немає. Слова «атмосфера» і «атмосфера» дуже близько один одному, хоча не є синонімами. У той же час «два» і «багато» стоять поруч, що логічно. Але загальний зміст цього прикладу в тому, що результати, які будуть отримано дуже сильно залежать від корпусу, над яким ведеться робота. [7]

Це приводить до наступної модифікації формули слово-документ – формулу TF-IDF (6). Ця абревіатура означає “term frequency – inverse document frequency”.

$$TF - IDF(\omega, d, C) = \frac{count(\omega,d)}{count(d)} * \log(\frac{\sum_{\acute{d}\in C} 1(\omega,\acute{d})}{|C|}), \quad (6)$$

де TF – це частота слова w у тексті d. IDF – логарифм оберненої частоти розповсюдженості слова w в корпусі C. Розповсюдженість – це відношення числа текстів, в яких зустрілося дане слово, до загального числа текстів у корпусі. За допомогою даної формули можна порівнювати тексти з меншим ризиком, ніж при використанні звичайних частот, оскільки вони беруть до уваги неважливі слова, такі як: «а», «таким чином», «і» і т.ін [6].

1.2.5 Нові підходи до обробки корпусів документів – модель Word2Vec

Описані вище підходи були (і залишаються) ефективними для областей, де кількість текстів низька і словник є обмеженим, хоча, як було описано вище, там теж є свої складності. Але з приходом в наше життя мережі Інтернет все стало одночасно складніше і простіше: у вільному доступі з'явилася безліч текстів, і ці тексти із змінним і розширюваним словником. З цим треба було щось робити, а раніше відомі моделі не могли впоратися з таким обсягом текстів. Кількість слів в англійській мові (якщо дуже грубо) становить мільйон – тільки матриця зустрічі пар слів буде 10^6x10^6. Таку матрицю навіть зараз дуже важко завантажити до оперативної пам'яті комп'ютерів, а, скажімо, 10 років тому про таке можна було не мріяти. Звичайно, було запропоновано безліч способів, що спрощують або розпаралелюють обробку таких матриць, але всі вони були паліативними. [6]

І тоді було запропонований вихід за принципом «той, хто нам заважає, той нам допоможе!». А саме, в 2013 році тоді мало кому відомий чеський аспірант Томаш Міколов запропонував свій підхід до word embedding, який він назвав word2vec . Його підхід заснований на іншій важливій гіпотезі, яку в науці прийнято називати гіпотезою локальності – «слова, які зустрічаються в однакових оточеннях, мають близькі значення». Поняття близькості в даному випадку є дуже широким, оскільки поруч можуть стояти тільки слова, що поєднуються. Наприклад, для нас звично словосполучення «стояча людина». А сказати «стояча ручка» ми не можемо, бо ці слова не поєднуються. [7]

Модель, запропонована Міколовим є дуже простою – ймовірність слова буде розрахована за його оточенням (контекстом). Тобто ми будемо вчити такі вектора слів, щоб ймовірність, яка привласнюється моделлю слову, була близька до ймовірності зустріти це слово в схожому оточенні в реальному тексті. [6]

$$P(w_o|w_c) = \frac{e^{s(w_o,w_c)}}{\sum_{w_i \in V} e^{s(w_i,w_c)}} \quad (7)$$

де w_o – вектор цільового слова, w_c – це деякий вектор контексту, обчислений (наприклад, шляхом усереднення) за рахунок векторів інших слів, що оточують необхідно слово. А $s(w_1, w_2)$ – це функція, що зіставляє двом векторам одне число. Наприклад, це може бути згадана вище косинусна міра подібності. [8]

Наведена формула (7) називається *softmax*, тобто «м'який максимум» (м'який – диференційований). Це потрібно для того, щоб наша модель могла навчатися за допомогою *back-propagation*, тобто процесу зворотного поширення помилки. [6]

Процес тренування влаштований таким чином: ми беремо послідовно (2k + 1) слів, слово в центрі є тим словом, яке має обчислюватися. А сусідні слова є контекстом довжини по k з кожного боку. Кожному слову в нашій моделі ставиться у відповідність унікальний вектор, який ми змінюємо в процесі навчання нашої моделі. В цілому, цей підхід називається CBOW - continuous bag of words. Неперервний (continuous) тому, що ми завантажуємо у нашу модель послідовно набори слів з тексту, а BoW бо порядок слів у контексті не важливий. [7]

Також Міколовим відразу був запропонований й інший підхід – протилежний до CBOW, який він назвав skip-gram, тобто «словосполучення з пропуском». Він відрізняється тим, що ми намагаємося з даного нам слова вгадати його контекст (точніше вектор контексту). У всьому іншому модель має відмінностей. [8]

Варто зазначити, що, незважаючи на те, що в модель прямо не було закладено жодної семантики, а лише статистичні властивості корпусів текстів, виявилось, що натренована модель word2vec може знаходити деякі семантичні властивості слів. Класичний приклад з роботи автора представлений на рис. 10. [7]

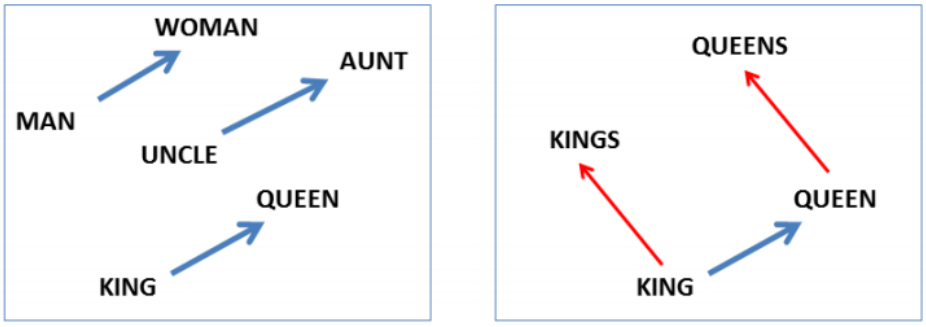

Рис. 1.14. Приклад знайдених семантичних зв'язків слів моделі word2vec

Слово «чоловік» відноситься до слова «жінка» так само, як слово «дядько» до слова «тітка», що для людей цілком природньо і зрозуміло, але в інших моделях досягнути такого ж відношення векторів можна тільки за допомогою спеціальних складних прийомів. Тут же - це відбувається природньо з самого корпусу текстів. До речі, крім семантичних зв'язків, установлюються і деякі синтаксичні (праворуч на рис. 10 показано співвідношення однини і множини). [17]

Насправді, за останній час було запропоновано багато покращень класичної моделі word2vec. Два самих розповсюджених будуть описані нижче. [7]

1.2.6 Оптимізація Word2Vec за допомогою Negative Sampling

В стандартній моделі CBoW, розглянутої вище, прогнозуються ймовірності слів та вони ж і оптимізуються. Функцією для оптимізації (в нашому випадку – мінімізації) служить дивіргенція Кульбака-Лейблера: [18]

$$KL(p||q) = \int p(x) log \frac{p(x)}{q(x)} dx, \tag{8}$$

де p(x) – розподілення ймовірностей слів, яке ми обчислюємо з корпусу, q(x) – розподілення, породжене нашою моделлю. Дивіргенція – це буквально «розходження», тобто міра, що показує наскільки одне розподілення не схоже на інше. Так як наші розподілення на словах дискретні, ми можемо замінити інтеграл у формулі на суму [18]:

$$KL(p||q) = \sum_{x \in V} p(x) log \frac{p(x)}{q(x)}. \tag{9}$$

Але виявилось, що оптимізувати цю формулу (9) досить складно. По-перше, це викликано тим, що q(x) розраховується за допомогою softmax за усім словником (англ. – 10^6 слів). А ще багато слів разом не зустрічається, тому більша частина обчислень буде надлишковою. Тому був запропонований елегантний обхідний шлях, який і отримав назву Negative Sampling. Суть цього підходу заключається в тому, що ми максимізуємо ймовірність зустрічі кожного слова в типовому контексті (тому,

який часто зустрічається в нашому корпусі) та одночасно мінімізуємо ймовірність зустрічі в нетиповому контексті (тому, який майже або зовсім не зустрічається). Це може бути описано наступною формулою [7]:

$$\mathrm{NegS}(w_0) = \sum_{i=1,\, x_i \sim D}^{i=k} -\log(1 + e^{s(x_i, w_o)}) + \sum_{j=1,\, x_j \sim \acute{D}}^{j=l} -\log(1 + e^{-s(x_j, w_o)}) \qquad (10)$$

де ***s(x,w)*** – така ж функція, що і в оригінальній формулі, а інші частини відрізняються. По-перше, необхідно звернути увагу, що формула тепер складається з двох основних частин: позитивної (***s(x,w)***) та негативної (***-s(x,w)***). Позитивна частина відповідає за типові контексти, а ***D*** – це розподілення сумісної кількості входжень слова w та інших слів. Негативна частина – це набір слів, які з нашим цільовим словом зустрічаються вкрай рідко. Цей набір породжується з розподілення D`, яке на практиці має вигляд звичайного нормального розподілення за всіма словами корпуса. Було показано, що така функція після оптимізації призводить до результату, аналогічному до стандартного softmax. [18]

1.2.7 Оптимізація Word2Vec за допомогою Hierarchical SoftMax

Також була спроба зайти і з іншого боку - можна не міняти вихідну формулу, а спробувати обрахувати сам *softmax* більш ефективно. Наприклад, використовуючи бінарне дерево. За всіма словами в словнику будується дерево Хаффмана. В отриманому дереві ***V*** слова розташовуються на листях дерева. [8]

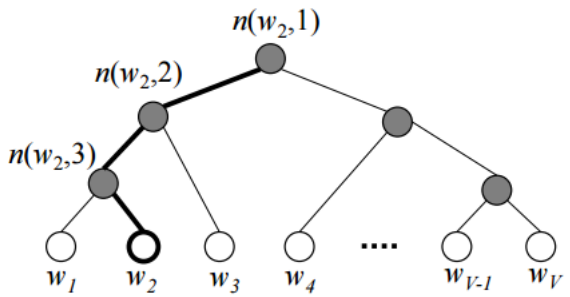

Рис. 1.15. Приклад побудови дерева Хаффмана

На рис. 11 зображено приклад такого бінарного дерева. Чорним кольором позначено шлях від кореня до слова w_2. Довжину шляху позначимо як L(w), а j-ту вершину на шляху до слова w позначимо n(w, j). Також можна довести те, що внутрішніх вершин $\boldsymbol{V-1}$. [8]

За допомогою ієрархічного softmax вектори $\boldsymbol{v_{n(w,j)}}$ обчислюються для $\boldsymbol{V-1}$ внутрішніх вершин. А ймовірність того, що слово $\boldsymbol{w}$ буде вихідним словом (в залежності від того, як будуються прогнозування: слово з контексту або задане слово за контекстом) обчислюється за формулою (11):

$$p(w = w_0) = \prod_{j=1}^{L(w)-1} \sigma\left(\left[n(w, j+1) = lch\left(n(w, j)\right)\right] v_{n(w,j)}^T u\right) \tag{11}$$

де $\boldsymbol{\sigma(x)}$ – функція softmax, [*true*] = 1, [*false*] = -1, $\boldsymbol{lch(n)}$ – лівий нащадок вершини n, $\boldsymbol{u = v_{wI}}$ у випадку використання метода *skip-gram* або $\boldsymbol{u = \frac{1}{h}\sum_{k=1}^{h} v_{wI,k}}$ (усереднений вектор контексту) у випадку використання методу *CBoW*. [8]

Зі сторони формула виглядає досить складною, але її легко зрозуміти, якщо уявити, що на кожному кроці ми можемо піти наліво або направо з ймовірностями показаними на формулах (12, 13) нижче [8]:

$$p(n, left) = \sigma(v_n^T u) \tag{12}$$

$$p(n, right) = 1 - p(n, left) = 1 - \sigma(v_n^T u) = \sigma(-v_n^T u) \tag{13}$$

Після цього на кожному кроці береться добуток ймовірностей ($L(w) - 1$ кроків) та ми виходимо на необхідну формулу (11).

При використанні звичайного softmax для підрахунку ймовірності слова, було необхідно обчислювати нормуючу суму за всіма словами зі словника ($O(V)$ операцій). Тепер ймовірність слова можна обчислити за допомогою послідовних обчислень, які потребують $O(log(V))$ операцій. [18]

1.3 Висновок

В розділі було розглянуто основний математичний апарат, який лежить за побудовою рекомендацій. Було розглянуто та проаналізовано структуру та методи побудови нейронних мереж, їх типи. Також було детально проаналізовано доречність використання багатошарових згорткових мереж, оскільки їх побудова потребує значних ресурсів.

Також під час дослідження було досить детально розглянуто алгоритм пошуку семантичних зв'язків між словами у вигляді побудови моделі Word2Vec з використанням алгоритмів SkipGram та CBoW. Після цього було наведено теоретичний опис можливостей покращення роботи даних алгоритмів, а саме алгоритми Negative Sampling та Hierarchical SoftMax.

Дані теоретичні відомості дозволяють тепер поставити інші завдання для дослідження такі як порівняння рекомендаційних систем відомих інтернет-сервісів, оскільки для розуміння структури їх рекомендаційних систем необхідно мати гарний структурованих базис знань з базових речей в методах глибокого навчання. Також даний теоретичний опис відкриває шлях до побудови практичних експериментів з використанням методів, описаних вище.

2 АНАЛІЗ СТРУКТУРИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ ВІДОМИХ ІНТЕРНЕТ-СЕРВІСІВ

2.1 Побудова Word Embeddings для оголошень Airbnb

2.1.1 Типи рекомендацій сервісу Airbnb

Ринок Airbnb містить мільйони різноманітних виставлених для оренди квартир, які потенційні гості обирають за допомогою результатів пошуку, створених за допомогою складної математичної моделі машинного навчання, яка використовує понад сотню факторів, щоб вирішити, як оцінювати певний запис у списку результатів пошуку. Коли гість переглядає будинок, вони можуть продовжувати пошук двома способами: або повертаючись до попередніх результатів, або переглядаючи рекомендації, що відображаються знизу на основі даного оголошення. Разом із рейтингом пошукових запитів та подібними (рекомендованими) лістингами конверсію за запитом було збільшено до 99%. [9]

У цьому розділі будуть описані основні методи побудови векторного представлення для оголошень, яка була розроблена в Airbnb з метою вдосконалення рекомендацій для результатів пошуку та персоналізації в режимі реального часу. Вкладання (embedding) – це векторні представлення оголошень (будинків) Airbnb, отримані під час пошукових сеансів користувачів, що дозволяють виміряти подібність між списками. В них закодовано багато інформаційних полів оголошень, таких як місце розташування, ціна, тип оголошення, тип архітектури та стилю – всі вони використовують лише 32-бітні числа з плаваючою комою. В Airbnb стверджують, що такий підхід є дуже ефективним та корисним для будь-якої e-commerce системи.

2.1.2 Теоретичний огляд векторного представлення (embeddings)

У ряді прикладних NLP (Natural Language Processing) задач класичні методи для лінгвістичного моделювання, що представляють слова як багаторозмірні, розріджені

вектора, були замінені нейронними лінгвістичними моделями що вивчають векторні представленя слів. В цих моделях використовуються низькорозмірні репрезентації слів завдяки використанню нейронних мереж. Мережі навчаються напряму на основі порядку слів та їхньому спільному використанню, базуючись на припущенні, що слова часто з'являються разом у реченнях, тобто в них більша статистична залежність. З розвитком легко масштабованих та неперервних пакетів слів та Skip-gram моделей для вивчення представлення слів було виявлено, що embeddings (векторні представлення) моделей отримують найсучаснішу продуктивність при виконанні багатьох традиційних мовних задач після тренування на великих об'ємах даних. [9]

Нещодавно концепція векторного представлення була розширена на інші прикладні області окрім векторного представлення слів NLP домену. Дослідники з доменних зон веб-пошуку, інтернет-комерції та маркет-плейсів з'ясували, що подібно до використання векторного представлення слів в реченні як контексті, можна використати векторні представлення дій користувача використовуючи послідовність дій користувача як контекст. Приклади включають в себе навчання за контекстом товарів, на які натиснув чи купив користувач, або запитів та рекламних оголошень, на які натиснув користувач. Ці векторні представлення були згодом використані для різних рекомендацій в Інтернеті.

2.1.3 Векторні представлення рекламних оголошень

Припустимо, що нам даний набір даних сесій кліків користувача, отриманий від $\boldsymbol{N}$ користувачів, де кожна сесія $s = (L_1,\ldots,L_n) \in S$ визначається як безперервна послідовність $\boldsymbol{n}$ індексів оголошень, на які натиснув користувач. Нова сесія стартує щоразу, коли між двома кліками користувача є проміжок часу більший за 30 хвилин. Враховуючи цей набір даних, метою є навчання 32-мірного представлення реальних значень $v(L_i) \in R^{32}$ кожного унікального повідомлення $\boldsymbol{L_i}$, що подібні повідомлення знаходяться неподалік в даному векторному просторі.

Розмірність векторного представлення було виставлено як d = 32, оскільки було виявлено, що це добрий компроміс між автономною продуктивністю (яку буде

оглянуто в наступному підрозділі) та пам'яттю, необхідною для зберігання векторів в оперативній пам'яті на пошукових машинах для цілей підрахунку коефіцієнтів подібності в режимі реального часу. [9]

Існує декілька різних способів для навчання embeddings. Сфокусуємось на техніці, що називається «Негативна вибірка». Даний метод спочатку ініціалізує векторні представлення як випадкові вектори. Після цього необхідно їх оновити за допомогою методу стохастичного градієнтного спуску, використовуючи сесії пошуку користувачів на манер методу ковзного вікна. На кожному кроці вектор центрального повідомлення зміщуються ближче до векторів оголошень з позитивним контекстом: такі, на які натиснув той самий користувач до чи після центрального оголошення у вікні довжиною *m (m = 5)*. Такі дії зміщують це оголошення далі від оголошень з негативним контекстом: оголошення, обрані випадковим чином (оскільки, швидше за все, вони не пов'язані з центральним).

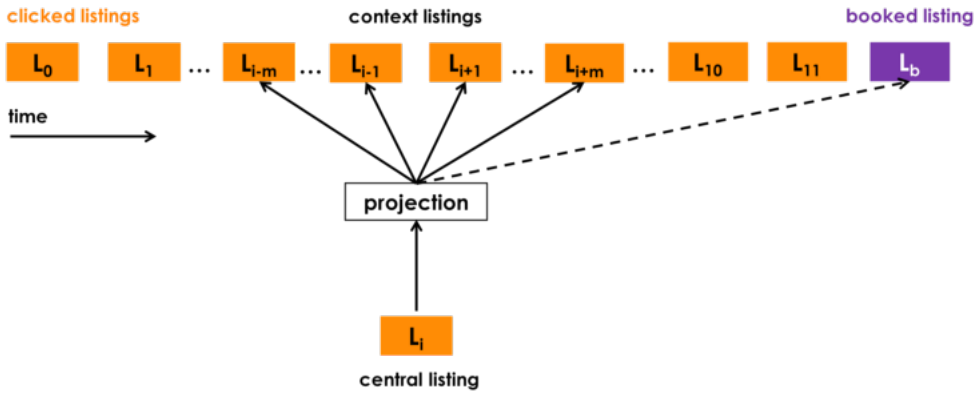

Рис. 2.1 – Приклад ранжування оголошень в результатах пошуку

Для наочності будуть пропущені деталі процедури навчання та подальший текст буде сфокусовано на поясненні деяких модифікацій для даного конкретного випадку:

- **Використання заброньованого оголошення як Глобального Контексту**: було використано сесії, що закінчуються бронюванням оголошення користувачем (фіолетове оголошення на рисунку), щоб пристосувати оптимізацію таким чином, щоб на кожному кроці ми прогнозували не тільки сусідні натиснуті оголошення, а також інші забронювані оголошення. З тим

як вікно рухається, деякі оголошення то попадають в контекст, то випадають з нього, тоді як забронюване оголошення завжди залишається в ньому як глобальний контекст (пунктирна лінія) і використовується для оновлення центрального векторного представлення оголошення.

- **Адаптація для Збірного Пошуку**: користувачі сайтів онлайн-бронювання зазвичай шукають житло лише на одному ринку, тобто в тій локації, де вони хочуть зупинитися. Як наслідок, для даного центрального оголошення позитивний контекст складається переважно з оголошень на тому ж ринку, тоді як перелік негативний контекст оголошень складається переважно з оголошень, які знаходяться на іншому ринку, бо їх було обрано випадковим чином з усього переліку оголошень. Було виявлено, що цей дисбаланс призводить до вивчення неоптимальних подібностей на ринку. Для вирішення цього питання запропоновано додати набір випадкових негативів оголошень ***Dmn***, обраних з ринку, де знаходиться центральне оголошення.

Беручи до уваги все, написане вище, фінальна ціль оптимізації (формула оптимізації забронюваного центрального контексту та негативних контекстів ринку) може бути сформована наступним чином:

$$\underset{\theta}{\operatorname{argmax}} \sum_{(l,c)\in\mathfrak{D}_p} \log\frac{1}{1+\ e^{-v`_c v_l}} + \sum_{(l,c)\in\mathfrak{D}_n} \log\frac{1}{1+\ e^{v`_c v_l}} + \log\frac{1}{1+\ e^{-v`_{l_b} v_l}} + \sum_{(l,m_n)\in\mathfrak{D}_{m_n}} \log\frac{1}{1+\ e^{v`_{m_n} v_l}} \tag{14}$$

де:

- ***l*** – це *центральне оголошення*, чий вектор v(l) проходить процедуру оновлення
- $\boldsymbol{D_p}$ – це позитивний набір пар *(l, c),* що представляє собою (*l - центральне оголошення, c - випадкове оголошення*) структуру даних т'юпл, чиї вектори підштовхуються в сторону один одного

- $\boldsymbol{D_n}$ – це негативний набір пар *(l, c),* що представляє собою (*l - центральне оголошення, c - випадкове оголошення*) структуру даних т'юпл, чиї вектори відштовхуються один від одного
- ***lb*** – це забронюване оголошення, що використовується як глобальний контекст і підштовхується в сторону центрального вектора оголошення
- $\boldsymbol{D_{mn}}$ – це негативний набір пар *(l, mn)* ринку, що представляє собою (*l - центральне оголошення, c - випадкове оголошення з того ж ринку*) структуру даних т'юпл, чиї вектори відштовхуються один від одного

Використовуючи описану процедуру оптимізації було отримано векторні представлення для 4.5 мільйонів рекламних оголошень квартир, використовуючи 800 мільйонів пошукових сесій користувачів, в результаті чого з'являються високоякісні репрезентації рекламних оголошень. [9]

2.1.4 Проблема холодного старту для векторних представлень

Щодня користувачі додають нові рекламні оголошення про доступні квартири на Airbnb. На цьому етапі ці оголошення ще не мають векторного представлення, оскільки вони не були представлені в тестових даних. Щоб створити embeddings для нового оголошення, зазвичай знаходяться 3 географічно найближчі квартири, які мають векторні представлення, і мають той самий тип оголошення та ціновий діапазон, що і новий запис, і обчислюється їх середній вектор. [9]

2.1.5 Яку додаткову інформацію надають векторні представлення?

Щоб оцінити, які характеристики записів були вкладені у векторні представлення, розглянемо їх кількома способами. По-перше, щоб оцінити, чи закодовано географічну подібність, обрахуємо кластеризацію ***k***-найближчих сусідів на отриманих векторних представленнях. Рис. 1, де показано, що в Каліфорнії виявлено 100 кластерів, підтверджує, що оголошення зі схожих місць об'єднуються разом. Далі було оцінено середні геометричні коефіцієнти подібності між оголошеннями різних типів (Entire Home, Private Room, Shared Room) та ціновими діапазонами. Було підтверджено, що ці коефіцієнти схожості типів набагато вище між оголошеннями того ж самого типу, ніж між оголошеннями різних типів та фактором

цінових діапазонів. Отже, можна зробити висновок, що ці дві характеристики оголошення добре кодуються у векторні представлення.

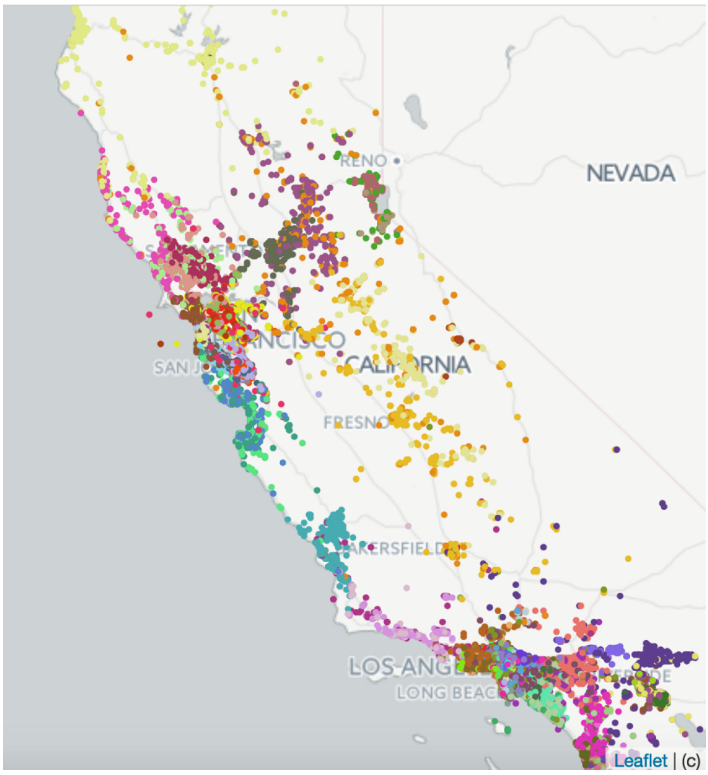

Рис. 2.2. Кластеризована мапа усіх оголошень Каліфорнії

Незважаючи на те, що деякі характеристики оголошення, такі як ціна, не потребують навчання, оскільки їх можна витягти із переліку метаданих, інші характеристики оголошення, такі як архітектура, стиль та атмосфера, набагато складніше витягати у формі функцій запису. Щоб оцінити ці характеристики та мати можливість проводити швидкі та прості дослідження в векторному просторі, було створено спеціальний інструмент внутрішнього аналізу подібності. [9]

2.1.6 Автономна оцінка векторних представлень оголошень

Перш ніж протестувати векторні представлення в рекомендаціях із реальним пошуковим трафіком, було проведено кілька тестів в автономному режимі. Також ці тести було використано для порівняння декількох різних навчальних елементів низькорозмірного векторного простору для швидкого прийняття рішень щодо розмірності цього представлення, різних ідей щодо модифікацій алгоритмів, побудови навчальних даних, вибору гіперпараметрів тощо.

Один із способів оцінки отриманих векторних представлень полягає в тому, щоб перевірити, наскільки добре вони рекомендують оголошення, які користувач забронює, на основі останнього кліку. [9]

Давайте припустимо, що нам надано нещодавно переглянуте оголошення та списки кандидатів, які мають бути відранжовані, що включають в себе оголошення, яке користувач нарешті забронював. Розраховуючи геометричні коефіцієнти подібності між векторними представленнями переглянутого оголошення та списками кандидатів, можна оцінювати кандидатів і спостерігати за рейтингом позиції забронюваного оголошення.

На рисунку нижче показано результати однієї такої оцінки, де оголошення в пошуковому записі були відранжовані на основі коефіцієнту подібності в просторі векторних представлень. Рейтинг кожного оголошення у списку усереднений за кожний клік (беруться до уваги ще 17 кліків до останнього кліку, що призвів до бронювання). [9]

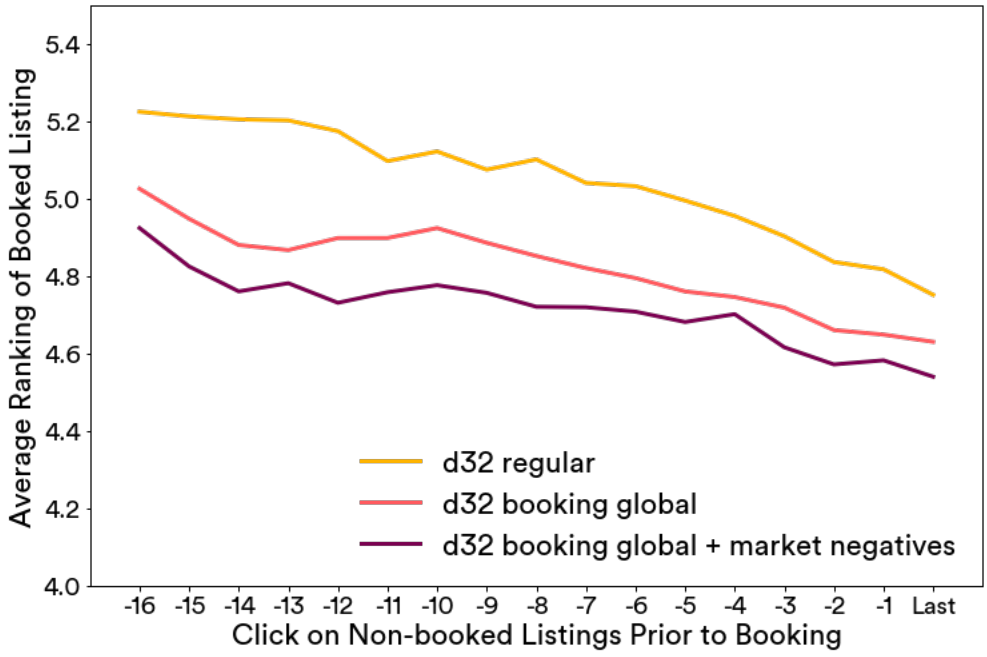

Рис. 2.3. Відношення рейтингу забронюваних оголошень до кліків на незабронювані оголошення

Було порівняно кілька версій ***алгоритму d32*** побудови векторних представлень (embeddings) в 32-мірному просторі:

1) *Алгоритм **d32 regular***, що пройшов навчання без будь-яких модифікацій оригінального алгоритму побудови векторних представлень;
2) *Алгоритм **d32 booking global** із забронюваними оголошеннями* як глобальним контекстом;
3) *Алгоритм **d32 booking global + market negatives**, де використані заброньовані оголошення та негативний контекст* (ринок), який було використано для побудови векторних представлень, що пройшли навчання з комбінованим забронюваним оголошенням як загальним та явними негативними прикладами оголошень з того ж ринку (контексту).

Завдяки оцінці, що полягає в підрахунку найнижчого середнього рейтингу забронюваного оголошення, можна заключити що ***d32 booking global + market negatives*** алгоритм має кращу продуктивність за два інших.

2.1.7 Схожі оголошення використовуючи векторні представлення

Кожне повідомлення на Airbnb містить в собі карусель зі схожими оголошеннями, на яких рекомендовані приміщення схожі на дане та вільні у заданий проміжок часу.

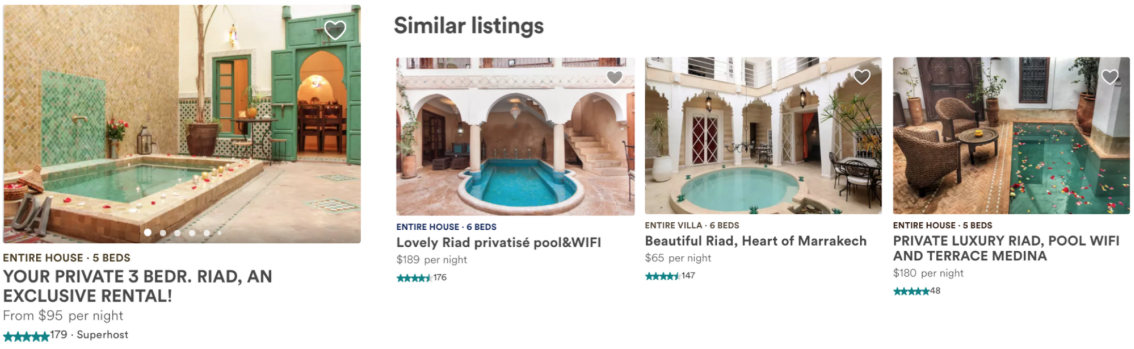

Рис. 2.4. Приклад каруселі схожих оголошень приміщень

На момент тестування алгоритму побудови векторних представлень існуючий алгоритм для пошуку схожих оголошень складався з виклику нашої основної моделі Ранжування результатів пошуку для тої ж локації як і у даного оголошення, а також з урахуванням фільтрації за ціною та типом оголошення.

Було проведено А/Б тестування, де порівнювалися існуючий алгоритм пошуку схожих оголошень із рішенням на основі побудови векторних представлень, в якому схожі оголошення отримувалися за рахунок пошуку *k-найближчих сусідів* у векторному просторі оголошень. Якщо більш точно, то з урахуванням навчених векторних представлень, схожі оголошення для даного оголошення ***l*** знаходилися після підрахування геометричного коефіцієнту схожості між його вектором ***v(l)*** та векторами $v(l_j)$ з усіх оголошень даного ринку оголошень, що доступні для наданого діапазону дат (якщо його виставлено). 12 оголошень із найбільшим коефіцієнтом відображаються в результатах каруселі як схожі. [9]

А/Б тестування показало, що рішення на базі побудови векторних представлень призвело до збільшення конверсії каруселі схожих повідомлень на 21% та на 4.9% більше гостей, що використовували карусель для оголошень закінчували сесію бронюванням.

2.1.8 Персоналізація пошуку в режимі реального часу з використанням Embeddings

До цього часу було показано, що векторні представлення можуть бути використані для ефективного обчислення подібності між оголошеннями. Наступна ідея полягає в тому, щоб використати цю можливість в алгоритмі ранжування результатів пошуку в режимі реального часу для персоналізації пошукових сесій користувачів, де ціллю є показати гостю більше оголошень, схожих на ті, що, на думку інженерів Airbnb, сподобалися йому з моменту початку пошукової сесії, та менше тих, що не сподобалися. [9]

Щоб досягти цього, необхідно збирати та тримати в пам’яті (використовуючи Kafka) в режимі реального часу 2 набори короткочасної історії подій:

1) ***Hc***: набір ідентифікаторів оголошень, на які натискав користувач протягом 2 тижнів.
2) ***Hs***: набір ідентифікаторів, що користувач пропустив за останні 2 тижні. За допомогою нього ранг пропущенні оголошень буде зменшено, оскільки до цього вони займали високі позиції в пошуку.

Після цього кожного разу, коли користувач користується пошуком, обчислюються 2 міри для кожного оголошення-кандидата $\boldsymbol{l_i}$ в результатах пошуку:

- ***EmbClickSim***: схожість між векторним представленням оголошення-кандидата та векторними представленням оголошень, на які користувач натиснув (з набору *Hc*).

$$EmbClickSim(l_i, H_c) = \max_{m \in M} \cos\left(v_{l_i}, \sum_{l_h \in m, l_h \in H_c} v_{l_h}\right) \quad (15)$$

Зокрема, необхідно підрахувати міру подібності між центроїдами на рівні ринку з набору даних ***Hc*** та обрати максимальний коефіцієнт схожості. Наприклад, якщо в ***Hc*** входять оголошення з Нью-Йорку та Лос-Анджелесу, векторні представлення оголошень кожного з цих двох ринків будуть усереднені, щоб сформувати нові ринкові центроїди.

- ***EmbSkipSim***: схожість між векторним представленням оголошення-кандидата та векторними представленням оголошень, які користувач пропустив (з набору *Hs*).

$$EmbSkipSim(l_i, H_s) = \max_{m \in M} \cos\left(v_{l_i}, \sum_{l_h \in m, l_h \in H_s} v_{l_h}\right) \quad (16)$$

Ці дві міри схожості було введено задля додаткового сигналу, який буде враховано моделлю машинного навчання ранжування результатів пошуку при ранжуванні оголошень-кандидатів на місце в результатах.

Це було імплементовано завдяки включенню цих двох мір схожості векторних представлень поряд з іншими критеріями ранжування результатів, щоб можна було сформувати новий маркований набір даних для тренування моделей, а потім навіть почати навчання нової моделі ранжування, яку буде протестовано згодом поряд з поточною за допомогою А/Б тестування.

Щоб підрахувати чи нова модель навчається використовувати нові міри схожості векторних представлень як було розраховано, було побудовано їх часткові

графіки на рисунку знизу. Ці графіки демонструють, що трапиться з рейтинком оголошення-кандидата, якщо будуть виправлені всі значення коефіцієнтів, крім поточної характеристики (тієї, що досліджується).

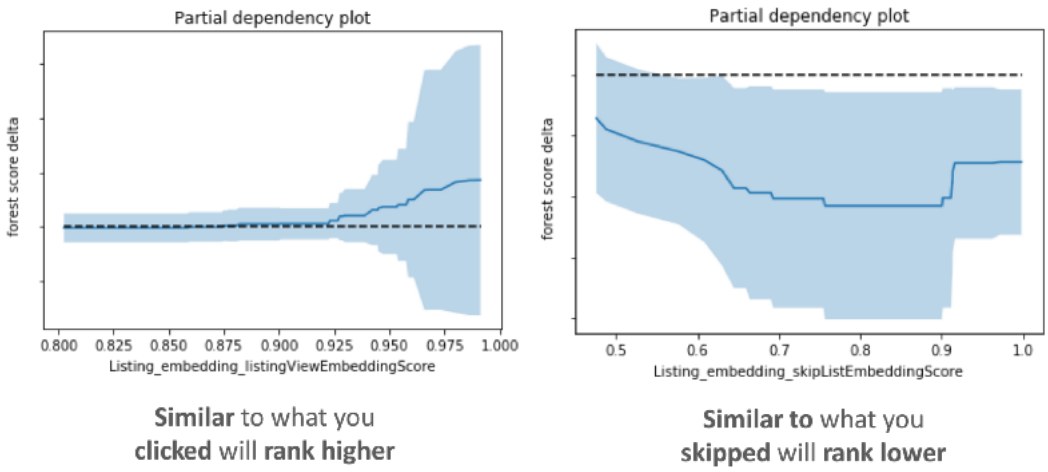

Рис. 2.5. Графіки дослідження ефективності EmbClickSim та EmbSkipSim

На лівому графіку показано, що більші значення EmbClickSim (схожі оголошення до оголошень, на які користувач нещодавно натискав) ведуть до більшого значення коефіцієнту рейтинга в результатах пошуку. На правому – більші значення EmbSkipSim (схожі оголошення до тих, які користувач пропустив – йому не вподобав) ведуть до зниження цього коефіцієнту. [9]

Спостереження з ділянок часткової залежності підтвердили, що особливості даних характеристик повністю співпадають з інтуїтивними очікуваннями від них. Окрім того, нові характеристики векторних представлень мають вищий рейтинг (більшу важливість) в моделі ранжування результатів пошуку та офлайн-тести показали значний рост продуктивності метрик на наборах даних, де ці характеристики були додані до моделі. Цього було достатньо, щоб провести вдалий онлайн-експеримент в режимі реального часу, що призвело до запуску цих характеристик на сайті в кінці 2017 року.

2.2 Глибокі нейронні мережі для рекомендацій сервісу YouTube

YouTube - це найбільша у світі платформа для створення, обміну та виявлення відеоматеріалів. Рекомендації YouTube несуть відповідальність за те, щоб допомогти

більше мільярду користувачів знаходити персоналізований контент із постійно зростаючої частини відео. У цьому документі ми зосередимося на величезному впливі глибокого навчання на відеорекомендації YouTube. На рисунку 9 ілюструються рекомендації на домашній сторінці мобільного додатка YouTube. [10]

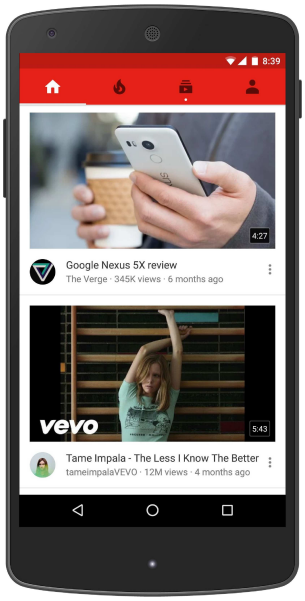

Рис. 2.6 - Домашня сторінка з рекомендаціями мобільного додатку Youtube

2.2.1 Загальна структура рекомендаційної системи YouTube

Загальна структура нашої системи рекомендацій наведена на рисунку 10. Система складається з двох нейронних мереж: одна для генерації кандидатів і одна для ранжування результатів пошуку кандидатів. Мережа з пошуку кандидатів бере події з історії активності користувача YouTube, і отримує невелику підмножину (сотні) відео з великого корпусу. Ці кандидати, як правило, мають відношення до користувача з високою точністю. Мережа для створення кандидатів забезпечує широку персоналізацію за допомогою спільної фільтрації. Подібність між користувачами виражається у вигляді грубих функцій, таких як ідентифікатори відео, токени пошукового запиту та демографічні показники. [10]

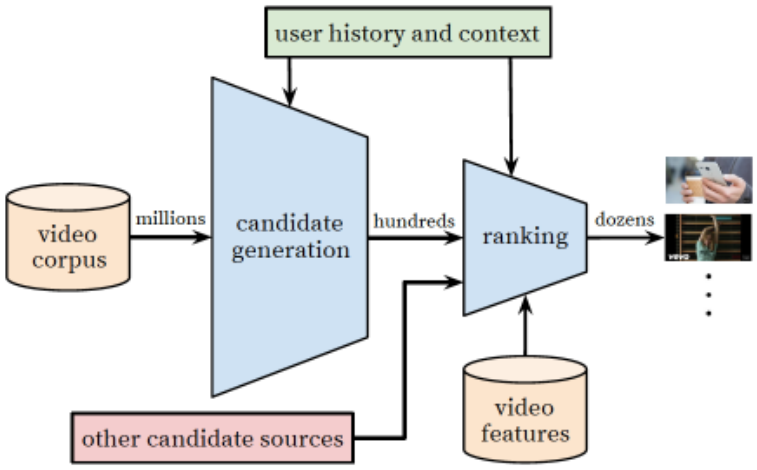

Рис. 2.7 - Загальна структура рекомендаційної системи YouTube

2.2.2 Структура мережі з генерації кандидатів для ранжування

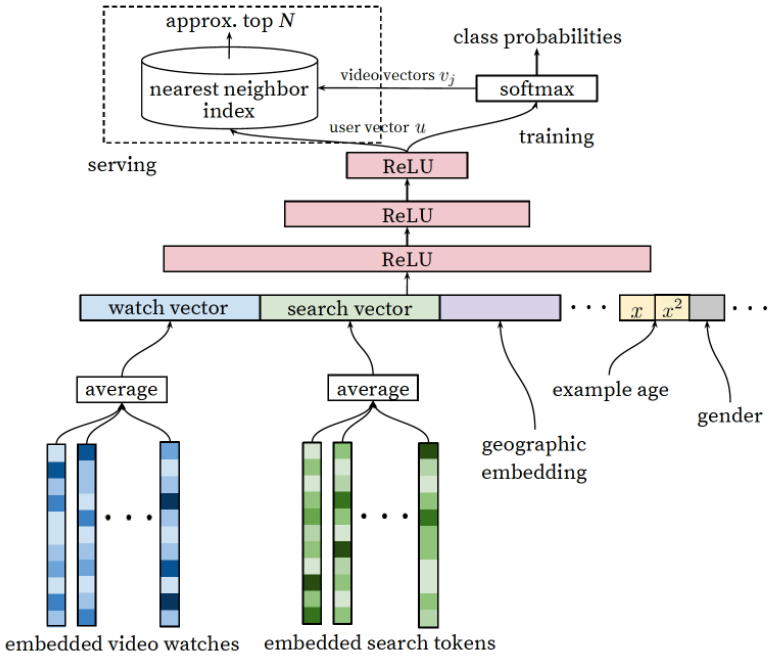

Рис. 2.8 - Структура мережі з генерацією кандидатів для ранжування

Архітектура моделі створення глибоких кандидатів має в середині вбудовані розріджені показники, об'єднані із щільними фічами. Згортки усереднюються до об'єднання, щоб перетворювати змішані розміри мішків розріджених ідентифікаторів у вектори з фіксованою шириною, придатні для введення в прихований шар. Всі приховані шари повністю підключені. Під час навчання втрати крос-ентропії зводиться до градієнтного спуску на виході з вибірки softmax. При обслуговуванні

приблизний пошук найближчого сусіда виконується для створення сотень рекомендацій щодо відео-кандидатів. [10]

2.3 Висновок

В даному розділі було розглянуто та проаналізовано рекомендаційні системи піонерів машинного навчання – сервісів Airbnb та YouTube. Сервіс YouTube виявився доволі консервативним: незважаючи на розповсюдженість ідей з приводу покращення роботи рекомендаційних систем та побудови зовсім нових рішень, компанія Google все ще має трохи застарілі алгоритми побудови ранжованого списку рекомендацій – згорткові нейронні мережі з використанням ReLU. Але як можна побачити з дослідження, вони почали використовувати сесії користувачів для побудови точніших рекомендацій.

Сервіс Airbnb навпаки намагається використовувати найсучасніші ідеї та технології в побудові рекомендацій. Звичайно, він може собі дозволити таке, оскільки розміри даного сервісну значно менші за масштаби YouTube (>1,000,000,000 відео). Тому, ідею використання моделі Word2Vec в рекомендаціях було успішно апробовано для текстових даних.

3 РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ВІДЕО-СЕРВІСУ

В наш час існує безліч інструментів для побудови нейронних мереж та проведення різних експериментів в сфері машинного навчання. Основні фреймворки написані та працюють за допомогою мови програмування Python – це PyTorch, TensorFlow, Keras, Caffe та Theano. Звичайно існують й інші. Серед цих інструментів можна виділити ті, що були створені крупними гравцями в секції глибокого навчання – це PyTorch (Facebook) та TensorFlow (Google). Тож необхідно зробити вибір на користь одного з них.

3.1 Порівняння основних засобів розробки глибокого навчання

В цьому розділі будуть розглянуто основні подібності та відмінності бібліотек PyTorch та TensorFlow.

Отже, TensorFlow було розроблено в компанії Google Brain. Ця бібліотека дуже активно використовується в Google як при дослідженнях, так і для production-потреб сервісів. Ця бібліотека знаходиться у відкритому доступі та розроблюється усією спільнотою. [11]

PyTorch – це відгалудження фреймворку Torch, створеного за допомогою мови lua. Ця бібліотека також широко використовується для різних експериментів у розробників та, зокрема, й у Facebook. Цей фреймворк не просто звичайний набір обгорток навколо основної версії. Його було повністю переписано з нуля, щоб операції виконувалися швидко та нативно. [12]

3.1.1 Deployment бібліотек PyTorch та TensorFlow

Звичайно, для невеликих додатків (вбудованих у Flask-додаток), де розгортання відбувається на сервері вручну, обидва фреймворки підходять, оскільки завантажити усі необхідні залежності не займає велику кількість часу.

Але, коли мова йде про розгортання на мобільних пристроях або у великих системах (частина з використанням цієї системи вбудована в основну), краще

використовувати TensorFlow. Звичайно, обидва фреймворки надзвичайно важко розгорнути та підтримувати для великих і складних систем, але при використанні TensorFlow не доведеться переписувати повністю частину зі статистичним виводом вашої моделі на Java або C++. Також була розроблена спеціальна обгортка – TensorFlow Serving – для таких «складних» розгортувань додатків. [13]

3.1.2 Data Loading бібліотек PyTorch та TensorFlow

Коли необхідна швидка робота з даними (завантаження, попередня обробка), то в PyTorch архітектура API краща. Інтерфейси для роботи вказані напряму в датасеті, зразках та завантажувачах даних. Завантажувач даних приймає набір даних і зразок даних та одразу будує ітератор над набором даних згідно до інструкцій зразка даних. Звичайно, завантаження даних можна розпаралелити, якщо передати параметр ***num_workers*** до завантажувача даних. [14]

В API TensorFlow нема досить корисних методів для завантаження даних (зчитувачів, черг, раннерів черг). Також увесь код, який необхідно написати, щобр розпаралелити завантаження даних, не є простим. Також API само по собі менш багатослівне, зрозуміле та його вивчити досить складно. Отже, можна побачити, що PyTorch зручніше користуватися. [13]

3.1.3 Порівняння бібліотек PyTorch та TensorFlow на прикладі відновлення вихідної функції

Для того, щоб практично порівняти роботу двох бібліотек, немає нічого кращого за вирішення конкретної задачі, а саме – відновлення вихідної функції. Отже, нехай маємо функцію $f(x) = x^{\phi}$. При заданих параметрах ***x*** та ***f(x)*** необхідно відновити степінь ***phi***. [14]

Для чистоти експерименту для розв'язку завдання побудуємо реалізацію градієнтного спуску вручну, тобто не будемо використовувати готові пакетні методи для даної задачі. Вже на цьому етапі в PyTorch вже є модуль ***optimize***, який має оптимізовані пакети моделі.[13]

Отже, на PyTorch реалізація буде виглядати наступним чином [15]:

```
def rmse(y, y_hat):
    """Підрахунок середньоквадратичної похибки"""
    return torch.sqrt(torch.mean((y - y_hat).pow(2)))

def forward(x, e):
    """Прямий хід нашої моделі"""
    return x.pow(e.repeat(x.size(0)))

# Задамо глобальні параметри
n = 1000 # кількість прикладів
learning_rate = 5e-10

# Побудова моделі
x = Variable(torch.rand(n) * 10, requires_grad=False)
y = forward(x, exp)

# Параметри моделі
exp = Variable(torch.FloatTensor([2.0]), requires_grad=False)
exp_hat = Variable(torch.FloatTensor([4]), requires_grad=True)

# Optimizer
opt = torch.optim.SGD([exp_hat], lr=learning_rate, momentum=0.9)

loss_history = []
exp_history = []

# Цикл тренування моделі
for i in range(0, 10000):
    opt.zero_grad()
    print("Iteration %d" % i)

    # Підрахунок поточного значення степеню
    y_hat = forward(x, exp_hat)

    # Обчислимо похибку
    loss = rmse(y, y_hat)

    # Запишемо інформацію для побудови графіків
    loss_history.append(loss.data[0])
    exp_history.append(y_hat.data[0])

    # Оновимо параметри моделі
    loss.backward()
    opt.step()

    print("loss = %s" % loss.data[0])
    print("exp = %s" % exp_hat.data[0])
```

Отже, як можна побачити з коду, його легко і швидко можна написати, навіть не маючи дуже глибоких знань бібліотеки PyTorch. Ініціалізація моделі та завантаження даних займають кілька строчок коду. Результати роботи даного коду можна побачити на рисунку 3.1.[14]

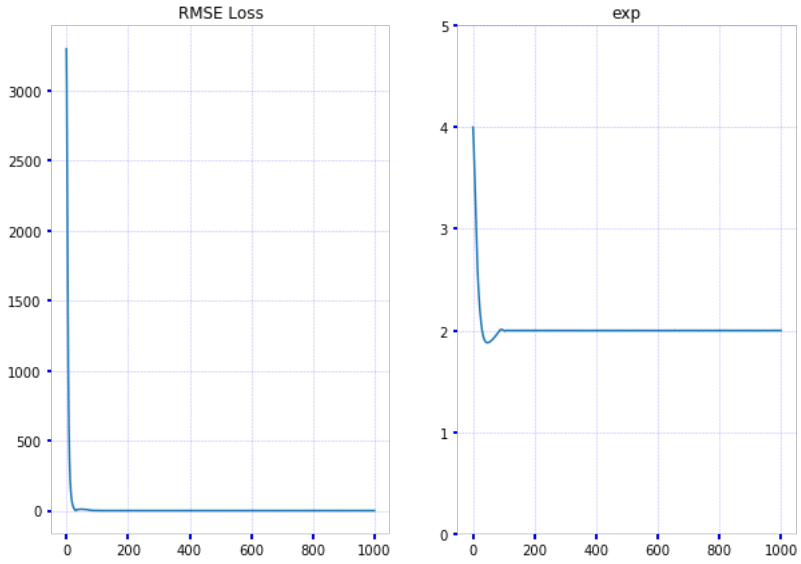

Рис. 3.1 – Loss-функція та графік пошуку степені на PyTorch

Отже, для відновлення поточної функції алгоритму, реалізованому на PyTorch, знадобилося близько 100 ітерацій. Інші виявилися надлишковими. Тепер необхідно виконати ті ж операції за допомогою фреймворку TensorFlow [15]:

```
def rmse(y, y_hat):
    """Підрахунок середньоквадратичної похибки"""
    return tf.sqrt(tf.reduce_mean(tf.square((y - y_hat))))

def forward(x, e):
    """Прямий хід нашої моделі"""
    # tensorflow автоматично оновлює модель
    # тому немає потреби у зміни форми e (reshape)
    return tf.pow(x, e)

n = 100 # кількість прикладів
learning_rate = 5e-6

# Заготовки для даних
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Параметри моделі
exp = tf.constant(2.0)
exp_hat = tf.Variable(4.0, name='exp_hat')

# Оголошення моделі
y_hat = forward(x, exp_hat)

# Оптимізатор
loss = rmse(y, y_hat)
opt = tf.train.GradientDescentOptimizer(learning_rate)


# We will run this operation to perform a single training step,
# Будемо запускати цю операцію, щоб зробити 1 тренувальний кроп
# аналог opt.step() в Pytorch.
```

```
# Виконання цієї операції також оновить параметри моделі
train_op = opt.minimize(loss)

# Генерація випадкових даних
x_train = np.random.rand(n) + 10
y_train = x_train ** 2

loss_history = []
exp_history = []

# Спочатку необхідно створити об'єкт TensorFlow Session
with tf.Session() as sess:

    # Ініціалізація усіх змінних
    tf.global_variables_initializer().run()

    # Тренувальний цикл
    for i in range(0, 500):
        print("Iteration %d" % i)
        # Запуск поточного тренувального кроку
        curr_loss, curr_exp, _ = sess.run([loss, exp_hat, train_op], feed_dict={
x: x_train, y: y_train})

        print("loss = %s" % curr_loss)
        print("exp = %s" % curr_exp)

        # Записи для побудови графіків
        loss_history.append(curr_loss)
        exp_history.append(curr_exp)
```

Результати роботи даного коду можна побачити на рисунку 3.2. Для відновлення даної функції phi знадобилося більше 300 ітерацій. [14]

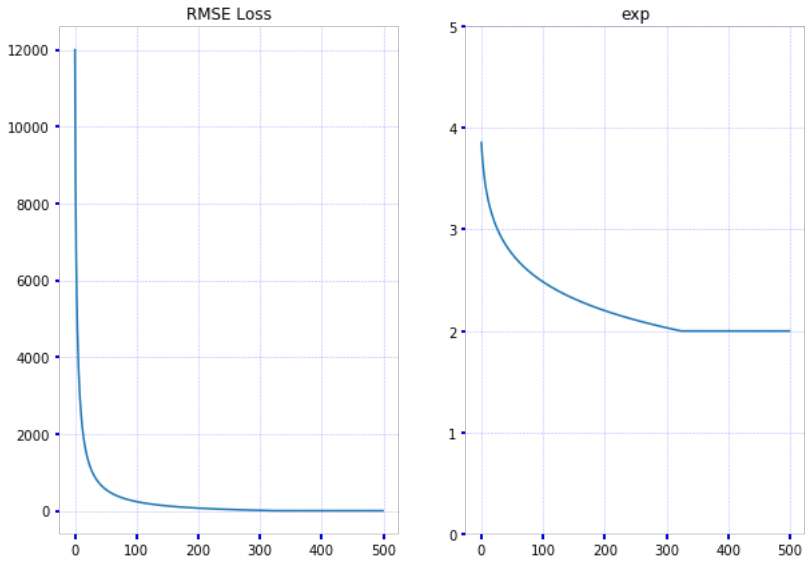

Рис. 3.2 – Loss-функція та графік пошуку степеня на TensorFlow

Отже, підводячи підсумок, можна побачити, що TensorFlow вже серйозна бібліотека для використання у складних та високо-навантажених системах, тому його використання завжди має бути виправданим, оскільки розробка коду та його подальша підтримка буде дуже ресурсоємна. У іншу чергу PyTorch підходить для досліджень краще, оскільки його API надзвичайно просте, елементарні операції оптимізовані та кількість усіляких зручних методів зростає кожний день. Тому у даній роботі буде використана саме бібліотека PyTorch. [14]

3.2 Побудова рекомендаційної системи

Отже, базуючись на теоретичних відомостях та інформації про рекомендаційні системи відомих інтернет-сервісів, можна побачити, що ідея побудови гарної рекомендаційної системи досить проста – використовуючи сесії користувачів, використати базову модель Word2Vec для тренування мережі та побудувати на базі даної моделі зручне API для користування рекомендаційною системою.

3.2.1 Підготовка даних

Для побудови нейронної мережі рекомендацій необхідно зібрати гарний датасет з сесіями користувачів, який можна буде використовувати для навчання нейронної мережі. Після проведення роботи з пошуку такого набору даних було вирішено використовувати набір даних з сервісу Kaggle.

Знайдений набір містить в собі анонімізовані активні сесії переходів користувачів за різними відео-файлами. Виглядає він наступним чином:

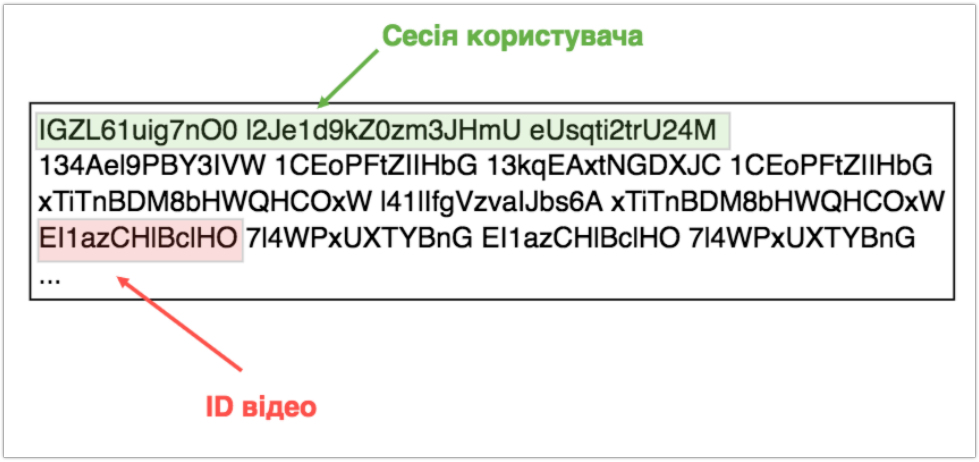

Рис. 3.3 – Структура даних підготовленого датасету

3.2.2 Архітектура нейронної мережі рекомендованої системи

Перед початком реалізації даної ідеї на PyTorch, необхідно спроектувати структуру нейронної мережі. Як вже зазначалося, за основу буде взята модель Word2Vec. Отже, нейронна мережа буде мати майже ідентичну структуру. Задача: отримати векторизовані представлення відео.

Почнемо з високо-рівневого розуміння того, що буде побудовано. Модель Word2Vec використовує звичайно згорткову нейронну мережу, де кожний прихований шар буде виконувати конкретну задачу, але ми не будемо використовувати цю нейронну мережу для задачі, яка буде їй поставлена. Замість цього необхідно вивчити ваги нейронів прихованого шару – саме вони будуть векторними представленями слів, які ми намагаємося вивчити.

3.2.2.1 Основна задача Word2Vec

На початку роботи зараз нам необхідно поставити «несправжнє» завдання для нейронної мережі, яку ми будемо будувати. Після цього, повернемося до того, як це завдання дає нам ті самі «векторні представлення слів», які ми шукаємо.

Отже, необхідно побудувати нейронну мережу, яка буде за заданим словом у середині речення (вхідне слово) шукати слова поряд та обирати одне випадковим чином. Мережа буде обчислювати ймовірність кожного слова зі словника стати цим «близьким» словом до того, яке ми обрали. Коли мова йде про «близькість», то мається на увазі параметр алгоритму «розмір вікна». Типовий розмір вікна може бути 5, який означає, що 5 слів до та 5 слів після даного будуть «близькими» до даного.[17]

Вихідні ймовірності будуть показувати наскільки близько кожне слово зі словника знаходиться до вхідного слова. Наприклад, якщо на вхід до навченої мережі подано слово “United”, то імовірність близькості слів до нього у “States” та “America” буде вищою, за ймовірність незв'язаних слів типу “watermelon” або “kangoo”.

Тренування нейронної мережі буде відбуватися наступним чином: на вхід будуть подаватися пари слів, знайдені в тренувальному корпусі документів. Приклад нижче (рис. 3.4) показує деякі з тренувальних прикладів (пари слів), які було взято із

малого вікна розміру 2 у реченні “The quick brown fox jumps over the lazy dog”. Вхідні слова підсвічено синім кольором. [17]

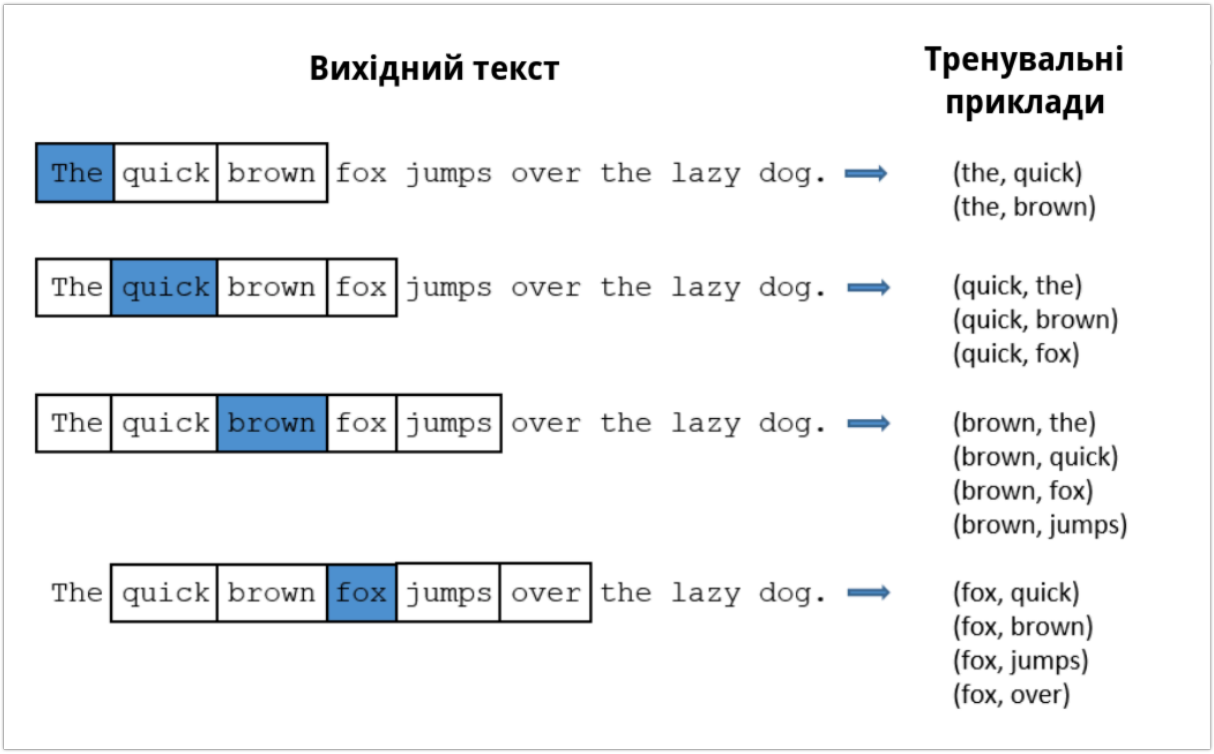

Рис. 3.4 – Приклад вхідних пар слів для тренування моделі Word2Vec

Нейронна мережа буде вивчати статистику кількості зустрічальності кожної з пар. Отже, для прикладу, нейронна мережа ймовірно буде мати набагато більше прикладів типу “United” - “States”, ніж “United” - “Roads”. Тож, після тренування якщо на вхід подано слово “United”, то на виході ймовірності для слів “States” та “America” будуть у рази вищими, ніж для слова “Roads”. [16]

3.2.2.2 Деталі моделі Word2Vec

По-перше, неможливо подати слово на вхід до нейронної мережі як String. Необхідно знайти спосіб репрезентації його у якийсь тип ідентифікатору. Щоб цього досягти необхідно спочатку побудувати словник з наших тренувальних документів. Нехай ми вже маємо словник із 10000 унікальних слів. [18]

Далі ми закодуємо кожне слово (наприклад “trees”) як one-hot вектор за допомогою алгоритму OHE (One-Hot Encoding). Цей вектор буде містити 10000

координат (по одній координаті для кожного слова зі словника). На позиції, що відповідає за слово “trees” буде стояти 1, на всіх інших – 0.

Виходом нейронної мережі буде звичайний вектор з 10000 координатами, в якому будуть міститися для кожного слова зі словника ймовірності того, що випадково взяте слово буде знаходитись поряд з даним.

Нижче на рис. 3.5 наведено архітектуру нейронної мережі:

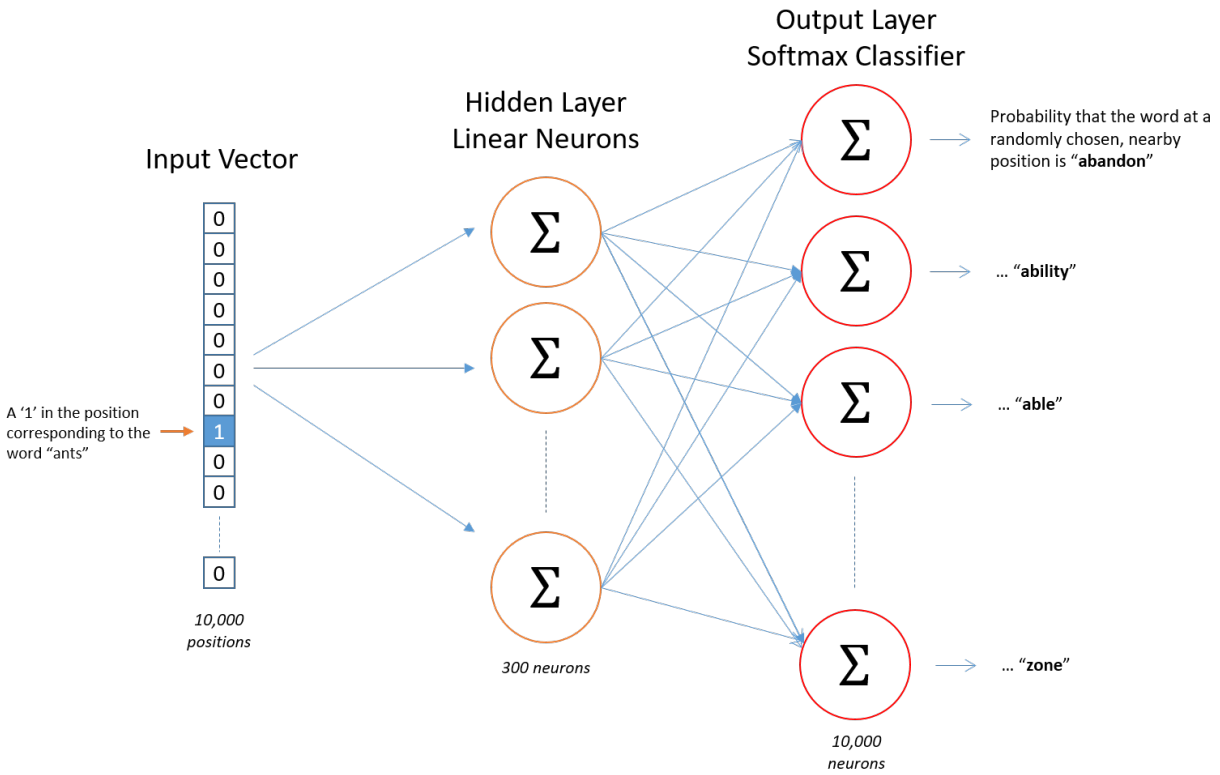

Рис. 3.5 – Архітектура нейронної мережі моделі Word2Vec

У прихованих шарах немає функції активізації, але вихідні нейрони використовують softmax, до якого повернемося пізніше. Коли відбувається тренування мережі на парах слів, вхідним словом буде one-hot вектор, що презентує це слово, а вихідним очікується також one-hot вектор, що представляє вихідне слово. Але при прогоні слова через мережу насправді ми отримуємо вихідний вектор, який є розподілом ймовірностей (набором чисел з плаваючою комою). [18]

3.2.2.3 Прихований шар моделі Word2Vec

Для нашого прикладу ми вирішили обрати навчаємо векторні представленя слів із 300 параметрами (features). Так, що прихований шар буде представлений матрицею вагів із 10000 рядків (для кожного слова у словнику) та 300 стовпців (по одному для кожного прихованого нейрона). Цифру 300 використав Google у першій натренованій моделі за даним алгоритмом. Ця кількість параметрів (300) є «гіпер-параметром», який необхідно налаштовувати у конкретній прикладній задачі. [17]

Якщо придивитися до матриці вагів прихованого шару (рис. 3.6), то можна зрозуміти що рядки цієї матриці і будуть векторними представленями слів.

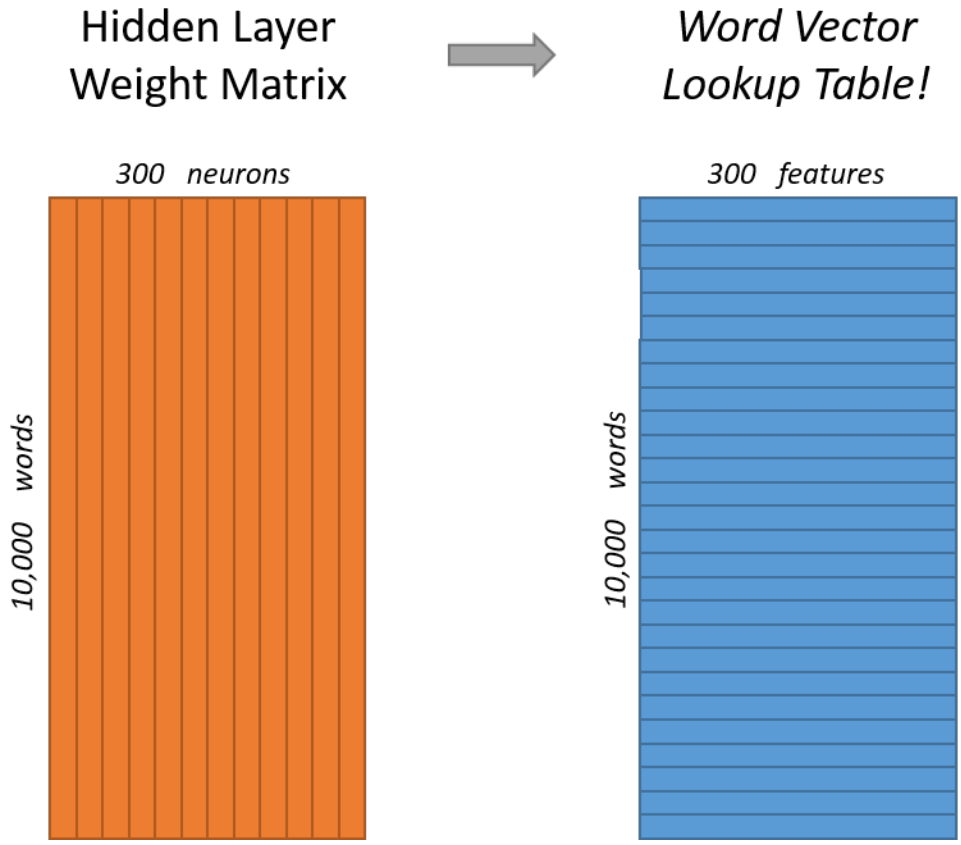

Рис. 3.6 – Матриця вагів прихованого шару

Кінцева мета цього представлення навчити нейронну мережу рахувати матрицю вагів прихованого шару – вихідний шар буде просто відкинутий після тренування. One-hot вектор майже на 100% складається з «0». Але перевага того, що при взятті добутку між 1x10,000 one-hot вектором та 10,000x300 матрицею, то ми

отримаємо лише рядок матриці вагів, який відповідає за «1». Це можна гарно побачити на рис. 3.7.

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Рис. 3.7 – Наочний приклад добутку one-hot vector та weight matrix

Це означає, що прихований шар виступає у реальності в якості таблиці пошуку. Виходом цього шару буде векторне представлення вхідного слова (“word embedding”).

3.2.2.4 Вихідний шар моделі Word2Vec

Вектор 1x300 для слова “trees” після обробки у прихованому шарі подається на вхід до вихідного шару. А вихідний шар є звичайним softmax regression класифікатором. Докладно про softmax написано в теоретичній частині. Коротко – кожний нейрон (на кожне слово у словнику) матиме значення «0» або «1», та сума всіх вихідних значень буде додана до «1». [18]

Якщо точніше, то кожний вихідний нейрон представлений наступною операцією: вектор вагів слова множиться на векторне представлення слова із прихованого шару, після чого над отриманим значенням виконуємо функцію ***exp(x)***. Після цього результат розділимо на суму результатів усіх 10,000 вихідних точок. Дану операцію гарно проілюстровано на рис. 3.8:

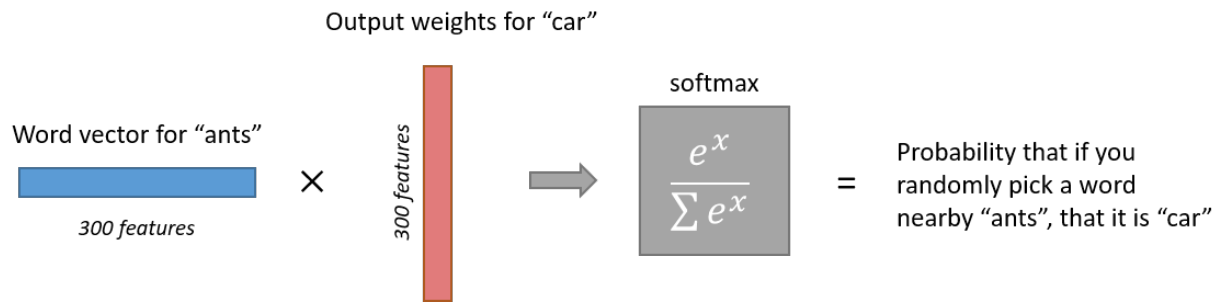

Рис. 3.8 – Операція на вихідному шарі нейронної мережі

Хочеться відмітити, що нейронна мережа не знає нічого про зміщення вихідного слова відносно вхідного. Вона не вивчає інший набір ймовірностей слова, що стоїть як перед вхідним, так і після нього. Щоб зрозуміти наведений тезис наведемо приклад. Нехай в навчальній вибірці (корпусі) кожне окреме входження слова “York” передує словом “New”. Тобто, базуючись лише на навчальній вибірці, існує 100% ймовірність того, що “New” буде в околиці слова “York”. Однак, якщо взяти, наприклад, 10 слів близьких до “York” та випадковим чином обрати одне з них, то ймовірність того, що це буде “New”, буде набагато нижче за 100% (ви маєте ймовірність обрати інше слово, відмінне від “New”). [17]

3.2.2.5 Підсумки моделі Word2Vec

Отже, якщо два різних слова мають дуже схожі «контексти» (слова, що дуже ймовірно з’являться коло них), тоді наша модель має на виході дуже схожі результати для цих двох слів. І одним зі способів нашою моделлю цього досягти – це видати на виході подібні векторні представлення слів. Тож, якщо два слова мають схожий контекст, то наша мережа намагатиметься видати схожі векторні представлення слів для них.

Та що ж таке для двох слів мати схожі контексти? Це означає, що такі слова як “intelligent” та “smart” матимуть дуже подібні контексти, оскільки вони є синонімами одне до одного. Або ж такі слова, як “engine” та “transmission”, швидше за все також матимуть схожі контексти. Також, як бонус, за даним алгоритмом побудови нейронної мережі, слова “tree” та “trees” матимуть схожі векторні представлення слів, бо вони також матимуть подібні контексти. [17]

3.2.3 Побудова рекомендаційної системи

Надана теоретична інформація Word2Vec та архітектура типової нейронної мережі для моделі Skipgram підводить до думки, що ID відео можна використовувати в якості слів, а сессії користувачів як речення або документи. Тому, теоретично, ця ідея має спрацювати схожим чином.

3.2.3.1 Побудова SkipGram моделі Videoid2Vec за допомогою PyTorch

Пропустивши блок з імпортом бібліотек і даних звичайна модель для побудови embeddings для рекомендації схожих відео матиме наступні параметри:

- ***emb_size***: Розмір векторного представлення.
- ***emb_dimention***: Розмірність простору embedding, зазвичай - 32.
- ***u_embedding***: Векторне представлення центрального відео.
- ***v_embedding***: Векторне представлення для сусідніх відео у сесії.

Ініціалізація моделі (показана на рис. 3.9) буде застосована для двох шарів нейронної мережі, що лежить в основі рекомендаційної системи. На цьому етапі будуть ініціалізовані ваги нейронів на прихованому шарі та на вихідному. Векторне представлення (***u_embedding***) центрального відео – це рівномірний розподіл в проміжку *[-0.5/em_size, 0.5/emb_size],* де вектори сусідніх відео (***v_embedding***) будуть заповнені нулями.

```
class SkipGramModel(nn.Module):
    """Skip gram model of gifid2vec.
    Attributes:
        emb_size: Embedding size.
        emb_dimention: Embedding dimention, usually 32.
        u_embedding: Embedding for center gif.
        v_embedding: Embedding for neibor gifs in session.
    """

    def __init__(self, emb_size, emb_dimension):
        """Initialize model parameters.
        Apply for two embedding layers.
        Initialize layer weight
        Args:
            emb_size: Embedding size.
            emb_dimention: Embedding dimention, typically 32.
        Returns:
            None
        """
        super(SkipGramModel, self).__init__()
        self.emb_size = emb_size
        self.emb_dimension = emb_dimension
        self.u_embeddings = nn.Embedding(emb_size, emb_dimension, sparse=True)
        self.v_embeddings = nn.Embedding(emb_size, emb_dimension, sparse=True)
        self.init_emb()

    def init_emb(self):
        """Initialize embedding weight like word2vec.
        The u_embedding is a uniform distribution in [-0.5/em_size, 0.5/emb_size], and the elements of v_embedding are zeroes.
        Returns:
            None
        """
        initrange = 0.5 / self.emb_dimension
        self.u_embeddings.weight.data.uniform_(-initrange, initrange)
        self.v_embeddings.weight.data.uniform_(-0, 0)
```

Рис. 3.9 – Ініціалізація моделі Videoid2Vec

Після ініціалізації знаходиться код для задання прямого ходу нейронної мережі — метод forward. На вхід він приймає 4 аргументи:

- ***pos_u***: список індексів центральних відео для позитивтих пар відео.
- ***pos_v***: список індексів сусідніх відео для позитивтих пар відео.
- ***neg_u***: список індексів центральних відео для негативних пар відео.
- ***neg_v***: список індексів сусідніх відео для негативних пар відео.

Так як за дизайном системи PyTorch необхідно, щоб усі входи даного методу були у форматі batch, всі аргументи є списками Video Id. Код прямого ходу представлений на рисунку 3.10.

```
def forward(self, pos_u, pos_v, neg_v):
    """Forward process.
    As pytorch designed, all variables must be batch format, so all input of this method is a list of gif id.
    Args:
        pos_u: list of center gif id indexes for positive gif pairs.
        pos_v: list of neibor gif id indexes positive gif pairs.
        neg_u: list of center gif id indexes negative gif pairs.
        neg_v: list of neibor gif id indexes negative gif pairs.
    Returns:
        Loss of this process, a pytorch variable.
    """
    emb_u = self.u_embeddings(pos_u)
    emb_v = self.v_embeddings(pos_v)
    score = torch.mul(emb_u, emb_v).squeeze()
    score = torch.sum(score, dim=1)
    score = F.logsigmoid(score)
    neg_emb_v = self.v_embeddings(neg_v)
    neg_score = torch.bmm(neg_emb_v, emb_u.unsqueeze(2)).squeeze()
    neg_score = F.logsigmoid(-1 * neg_score)
    return -1 * (torch.sum(score) + torch.sum(neg_score))
```

Рис. 3.10 – Код прямого ходу нейронної мережі

3.2.3.2 Побудова тренування нейронної мережі моделі SkipGram

Модель тренування представлена класом SkipGramTrainer. Для моделі тренування необхідо ініціалізувати вхідні дані. Код представлений на рисунку 3.11. На вхід до класу SkipGramTrainer подаються наступні аргументи:

- ***input_file_name***: Ім'я вхідного текстового файлу з даними. Кожний рядок містить ID відео, що були переглянуті користувачем під час сесії, розділені пробілами
- ***output_folder***: Ім'я вихідної директиви для фінальних embeddingsName of the final embedding folder.

- ***emb_dimention***: Розмірність векторного простору - зазвичай 32
- ***batch_size***: Кількість пар ID відео для одного прямого ходу
- ***window_size***: Розмір вікна для пошуку сусідніх відео
- ***iteration***: Контролер ітерації тренування
- ***initial_lr***: Ініціалізація коефіцієнту навчання
- ***min_count***: Мінімальна частота появ відео. Відео з нижчою частотою буде відфільтровано.

```
class SkipGramTrainer:
    def __init__(self,
                 input_file_name,
                 output_folder,
                 emb_dimension=32,
                 batch_size=512,
                 window_size=3,
                 iteration=5,
                 initial_lr=0.01,
                 min_count=5):
        """Initilize class parameters.
        Args:
            input_file_name: Name of a text data from file. Each line is a video ids sequence splited with space.
            output_folder: Name of the final embedding folder.
            emb_dimention: Embedding dimention, typically 32.
            batch_size: The count of video id pairs for one forward.
            window_size: Max skip length between videos.
            iteration: Control the multiple training iterations.
            initial_lr: Initial learning rate.
            min_count: The minimal video frequency, videos with lower frequency will be filtered.
        Returns:
            None.
        """
        self._data = SkipGramDatasetReader(input_file_name, min_count)
        self._output_folder = output_folder
        self._emb_size = len(self._data.video2id)
        self._emb_dimension = emb_dimension
        self._batch_size = batch_size
        self._window_size = window_size
        self._iteration = iteration
        self._initial_lr = initial_lr
        self._skip_gram_model = SkipGramModel(self._emb_size, self._emb_dimension)
        self._use_cuda = torch.cuda.is_available()
        if self._use_cuda:
            self._skip_gram_model.cuda()
        self._optimizer = optim.SGD(self._skip_gram_model.parameters(), lr=self._initial_lr)
```

Рис. 3.11 – Ініціалізація класу SkipGramTrainer

Процес тренування полягає в поділу вхідних даних на пари відео для тренування моделі. Після формування таких пар формуються набори (batches) відео для створення кількох епохів тренування, оскільки неможливо одночасно обробити велику кількість даних. Код тренування представлений на рисунку 3.12:

```python
def train(self):
    """Multiple training.
    Returns:
        None.
    """
    pair_count = self._data.evaluate_pair_count(self._window_size)
    batch_count = self._iteration * pair_count / self._batch_size
    process_bar = tqdm(range(int(batch_count)), smoothing=1.0)

    for i in process_bar:
        pos_pairs = self._data.get_batch_pairs(self._batch_size, self._window_size)
        neg_v = self._data.get_neg_v_neg_sampling(pos_pairs, 5)
        pos_u = [pair[0] for pair in pos_pairs]
        pos_v = [pair[1] for pair in pos_pairs]

        pos_u = Variable(torch.LongTensor(pos_u))
        pos_v = Variable(torch.LongTensor(pos_v))
        neg_v = Variable(torch.LongTensor(neg_v))
        if self._use_cuda:
            pos_u = pos_u.cuda()
            pos_v = pos_v.cuda()
            neg_v = neg_v.cuda()

        self._optimizer.zero_grad()
        loss = self._skip_gram_model.forward(pos_u, pos_v, neg_v)
        loss.backward()
        self._optimizer.step()

        process_bar.set_description(f'Loss: {loss.data[0]:.4f}, lr: {self._optimizer.param_groups[0]["lr"]:.6f}')
        if i * self._batch_size % 100000 == 0:
            lr = self._initial_lr * (1.0 - 1.0 * i / batch_count)
            for param_group in self._optimizer.param_groups:
                param_group['lr'] = lr
    self._save_embedding()
```

Рис. 3.12 – Код тренування моделі SkipGram (SkipGramTrainer)

Звичайно, як було сказано в розділі про Word2Vec модель, нас цікавить саме прихований шар. Тож після кожного епоху (epoch) ми будемо зберігати дані до так званого словника (не плутати зі словником для one-hot encoding). В ньому будуть зберігатися векторні представлення відео – тобто так званий словник у новому векторному просторі. Код заповнення словника представлений на рисунку 3.13.

```python
def _save_embedding(self):
    ensure_dir(self._output_folder)

    if self._use_cuda:
        embedding = self._skip_gram_model.u_embeddings.weight.cpu().data.numpy()
    else:
        embedding = self._skip_gram_model.u_embeddings.weight.data.numpy()

    video_to_vec = {}
    for video, index in self._data.video2id.items():
        video_to_vec[video] = embedding[index]

    with open(os.path.join(self._output_folder, 'vocabs'), 'wb') as output_file:
        pickle.dump(
            {
                'video_to_vec': video_to_vec,
                'index_to_video': self._data.video2id
            },
            output_file
        )

    del video_to_vec

    nms_index = nmslib.init(method='hnsw', space='cosinesimil')
    nms_index.addDataPointBatch(embedding)
    nms_index.createIndex()

    nmslib.saveIndex(nms_index, os.path.join(self._output_folder, 'nms_index'))
```

Рис. 3.13 – Побудова нового «словника» векторних представлень (video embeddings)

Звичайно, для запуску даного коду та ефективній роботі необхідно використовувати ресурси GPU, тому недивно, що для їх задіяння було використано бібліотеку CUDA. Також було застосовано код деяких оптимізаторів, який буде наведено у Додатку.

3.2.3.3 Побудова веб-додатку рекомендаційної системи

Отже, після опрацювання всіх вхідних даних ми побудували векторне представлення у 32-мірному просторі для кожного відео. Такий словник вже можна використовувати для побудови рекомендаційної системи. Звичайно, що її структура буде надзвичайно простою. На вхід буде подаватися відео з корпусу відео, а на виході ми матимемо 50 схожих відео. Схожі відео будуть обиратися з корпусу відео шляхом пошуку найближчого – існує безліч інструментів пакету PyTorch для цього.

Простий веб-додаток для наочної демонстрації ефективності експерименту було побудовано за допомогою фреймворку Flask (back-end) та React/Redux (front-end). Код для даної частини розробки рекомендаційної системи є тривіальним та не входить в суть дослідженя, тож його буде наведено в Додатку, а опис буде опущено. Результат побудови додатку можна побачити на рисунку 3.14.

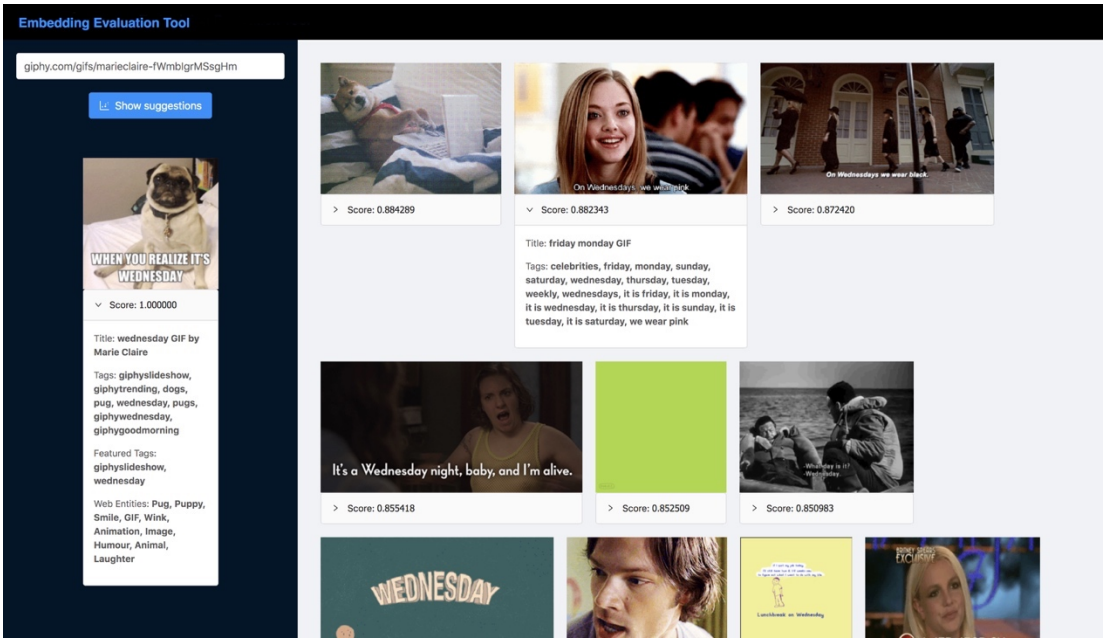

Рис. 3.14 – Приклад веб-додатку рекомендацій коротких відео

Як можна побачити з прикладу роботи програми, надано релевантні рекомендації. Центральне відео (вхідне) має назву «When you realize it’s Wednesday» та рекомендації сусідніх відео із “Wednesday Celebration”, що є більш ніж гарним результатом, оскільки вихідні відео є досить релевантними.

3.2.3.4 Результати роботи рекомендаційної системи

Після побудови додатку було проведено тестування системи. 90% корпусу відео було надано для тренувальних цілей, а 10% - для тестування. Це звичайний крок при побудові експериментів глибокого навчання. Графік на рисунку 3.15 демонструє результати тестування, а саме на якій позиції виявилися рекомендовані відео відносно тестового прикладу. Тобто, якщо позиція відео в рекомендації збігається з тестовою, то конкретна колонка, що відповідає за дану позицію, буде збільшена на 1. Якщо позиція >= 50 – останню колонку буде збільшено на 1.

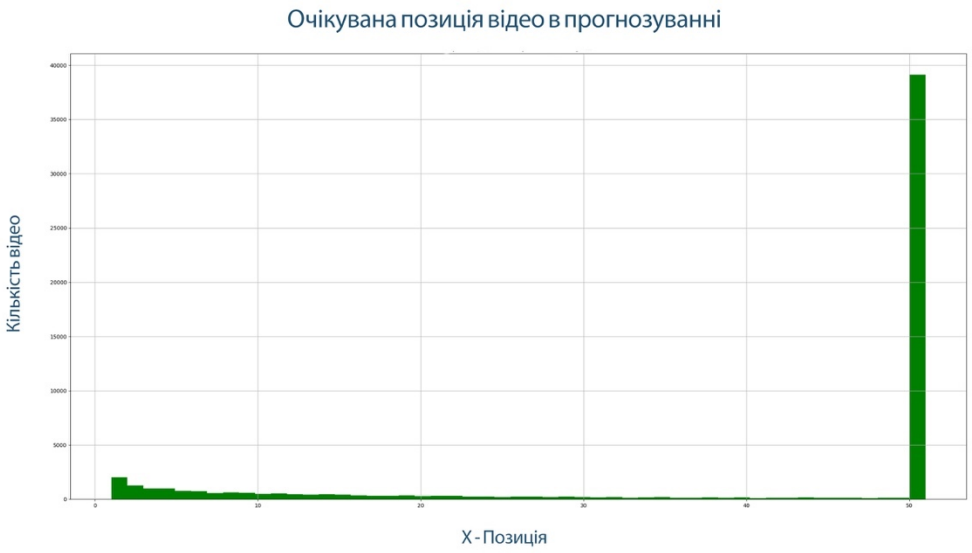

Рис. 3.15 – Графік очікуваної позиції відео в рекомендаціях

Як можна побачити з даного графіку ми отримали дуже гарний результат, оскільки дуже багато даних вдалося рекомендувати правильним чином. Звичайно, те, що остання колонка виявилася досить великою не є дуже добрим знаком. Але це лише є знаком того, що ми просто не вгадали позицію відео.

Наступний графік на рисунку 3.16 показує яким чином відео розподілені у корпусі документів. По осі абсцис наведені стовпці кількості зустрічань конкретного

відео в сесіях користувачів, а по осі ординат – кількість таких відео. Графік показує, що з даних, отриманих з датасету, дуже важко було б побудувати рекомендаційну систему базуючись лише на назвах та контенту – тож на таких даних дуже гарно спрацювала модель SkipGram, як можна побачити у Веб-додатку.

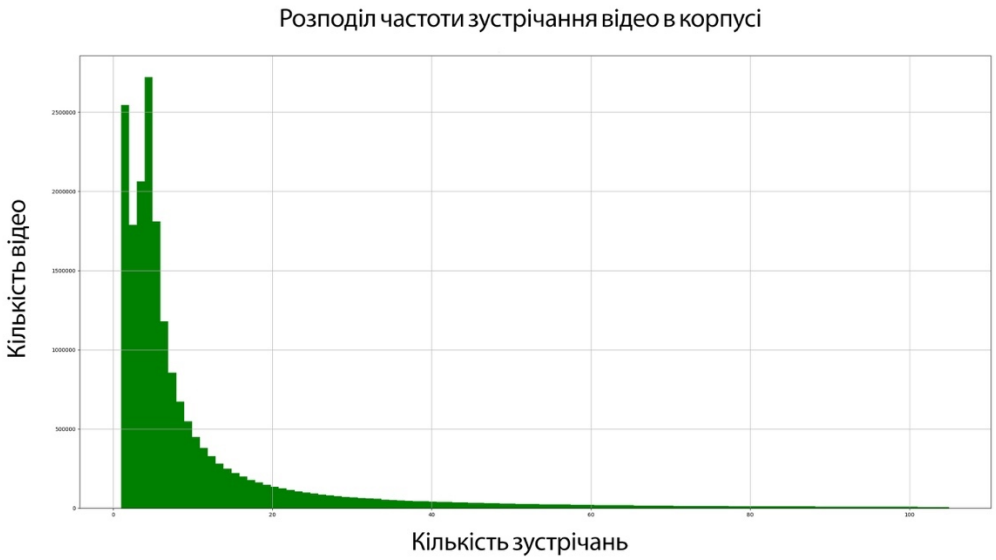

Рис. 3.16 – Розподіл відео у корпусі документів

3.2.4 Подальший розвиток дослідження рекомендаційних систем

Звичайно, як можна побачити, дана модель вже поводить себе належним чином — рекомендації є релевантними. Дану тему можна розвивати нескінчено, покращуючи якість рекомендацій. Для цього, по-перше, можна збільшити корпус відео та краще розпаралелити навчання. По-друге, можна використовувати більше даних, оскільки окрім ID відео можна використовувати як мета-інформацію, так і сам контент відео. Тоді задача з NLP алгоритмів буде ускладнена Computer Vision складовою, що, звичайно, збільшує час розробки, але, в теорії, може покращити якість рекомендацій.

3.3 Висновок

При реалізації алгоритмів глибокого навчання, а саме – побудові експериментів, використовується безліч фреймворків та бібліотек. Зараз існує 2 основних інструменти production-рішень для таких експериментів – PyTorch та TensorFlow. Після аналізу бібліотек прийшли до висновку, що PyTorch краще

підходить для побудови невеликих експериментів, є більш зручним для роботи з даними та краще виконує типові задачі такі, як реалізація алгоритму градієнтного спуску, побудова згорткових нейронних мереж тощо.

В даному розділі було детально проаналізовано алгоритм побудови моделі Word2Vec з інженерного боку для пошуку семантичних зв’язків між словами, оскільки він став ключовою ідеєю для побудови рекомендаційної системи. В результаті аналізу даної моделі було виявлено, що семантичні зв’язки між словами можна шукати за допомогою алгоритму SkipGram.

Також було побудовано рекомендаційну систему для великого корпусу відео на основі SkipGram моделі. Після побудови моделі було створено веб-додаток за допомогою Flask та Redux для наочної демонстрації результатів дослідження. З результатів роботи рекомендаційної системи було зроблено висновок, що система поводить себе коректно, рекомендації – релевантні. Тому можна використовувати поточне рішення для різного роду медіа-платформ для збільшення конверсії тощо.

4 РОЗРОБКА СТАРТАП-ПРОЕКТА «МІКРОСЕРВІС РЕКОМЕНДАЦІЙ ВІДЕО НА ОСНОВІ СЕСІЙ КОРИСТУВАЧІВ»

Розділ має на меті проведення маркетингового аналізу стартап проекту “Мікросервіс рекомендацій відео на основі сесій користувачів” задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження.

Метою розділу є формування інноваційного мислення, підприємницького духу та формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді розроблення концепції стартап-проекту “Мікросервіс рекомендацій відео на основі сесій користувачів” в умовах висококонкурентної ринкової економіки глобалізаційних процесів.

Опис стартап-проекту “Мікросервіс рекомендацій відео на основі сесій користувачів” наведено у Таблиці 4.1.

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Розробка мікросервісу, який буде рекомендувати відео на основі сесій користувачів, які будуть передаватися за допомогою API.	1. Використання як бібліотеки в проекті що використовує Flash/Django.	Використання як бібліотеки безпосередньо дозволяє швидко й гнучко оперувати інформацією, а також оберігає від помилок через неуважність
	2. Використання як сервісу через REST API.	REST API надає можливість використовувати систему як сервіс

Отже, проект “Мікросервіс рекомендацій відео на основі сесій користувачів” може бути використано як інструмент для рекомендацій відео за допомогою сесій користувачів, Таблиця 4.2.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

п/п	*Техніко-економічні характеристики ідеї*	*(потенційні) товари/концепції конкурентів*				*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Мій проект	*Конкурент 1*	*Конкурент2*	*Конкурент 3*			
.	Форма виконання	Веб-сервіс, бібліотека	Бібліотека	Веб-додаток	Веб-додаток			+
.	Кросплатформність	Так	Ні	Так	Так			+
.	Наявність використання безпосередньо в сервісі	Так	Так	Ні	Ні			+
.	Швидкість	Висока	Низька	Низька	Низька			+
.	Горизонтальне масштабування	Так	Ні	Ні	Ні			+
.	Можливість валідації повідомлення на коректність	-	+	+	+	+		

Наш сервіс-бібліотека надає дуже широкі можливості використання у всіх умовах. Але за рахунок того що основний акцент було зроблено на швидкість та зручність, постраждала можливість валідації документів з відео ID, але в подальшому, її можна буде легко додати. Отож, система є конкурентноспроможною.

4.1 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару) Таблиця 4.3.

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Побудова моделі векторних представлень відео	PyTorch	Наявна	Безкоштовна, доступна
2.	Створення інтерфейсу користувачів	React/Redux	Наявні	Безкоштовна, доступна
3.	REST API та рекомендаційної системи	Python Flask	Наявна	Безкоштовна, доступна

Обрані технології реалізації ідеї проекту: PyTorch через повну безкоштовність фреймворку та швидкість його з поміж конкурентів, наявність досвіду роботи розробників з даною технологією; Python Flask через простоту використання, наявність досвіду роботи розробників з даною технологією, безкоштовність та можливості для цього завдання; React/Redux через безкоштовність і наявність досвіду.

4.2 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів, Таблиця 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1.	Кількість головних гравців, од	5
2.	Загальний обсяг продаж, грн/ум.од	8000 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	R = (3000000 * 100) / (1000000 * 12) = 25%

Отже, було проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (табл. 4.5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1.	Необхідне програмне	Потенційними цільовими групами є дослідницькі	Цільова група займається дослідженнями	Рішення повинне бути придатним до інтеграції в інші

	забезпечення(REST API)	центри, університети та компанії, специфіка роботи яких потребує досліджувати методи глибокого навчання та шукати відео-рекомендації	або має обробляти дані сесій користувачів	більш складні системи, бути швидким та гнучким

Згідно проведеної характеристики потенційних клієнтів стартап-проекту випливає, що на ринку є затребуваним програмне забезпечення (REST API та бібліотеки) для побудови ранжованого списку рекомендацій і потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує досліджувати методи глибокого навчання та шукати відео-рекомендації.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. № 4.6-4.7). Фактори в таблиці подавати в порядку зменшення значущості.

Таблиця 4.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1.	Конкуренція	Вихід на ринок великої компанії	1. Вихід з ринку 2. Запропонувати великій компанії поглинути себе 3. Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу

			міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	1. Передбачити можливість додавання нового функціоналу до створюваного ПЗ
4.	Надходження на ринок альтернативних продуктів	Перехід користувачів нашого товару на інший продукт	Впровадження нового функціоналу, якого немає у конкурентів
5.	Уповільнення росту ринку	Скорочення користувачів продуктів, що тільки виходять на ринок	Інвестиції у впровадження ефективної реклами продукту

Отже, було проаналізовано фактори загроз ринкового впровадження проекту, серед яких: конкуренція, уповільнення росту ринку, зміна потреб користувачів, надходження на ринок альтернативних продуктів. Було також запропоновано можливі реакції компанії.

Таблиця 4.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1.	Стрімкий ріст попиту на інструменти побудови ранжованих списків рекомендацій	Наявність попиту на інструменти для побудови інтегрованих рекомендаційних систем	Змога запропонувати продукт більшій кількості потенційних користувачів
2.	Поява нових різонерів	Надання нового функціоналу для надання	Розробка нового функціоналу у вигляді нового HTTP запиту для

		результатів роботи нового запитів	надання користувачам результатів запитів
3.	Стрімке зростання росту ринку	Компаніям, що тільки виходять на ринок, буде простіше отримати клієнтів	Змога запропонувати продукт більшій кількості потенційних користувачів
4.	Обслуговування додаткових груп споживачів	Поява нових потенційних груп споживачів	Змога розширити продукт для подальшого впровадження у нові галузі
5.	Розширення асортименту можливих послуг	Поява нового функціоналу, що привабить нових користувачів	Розробка нового функціоналу, що є потребою певної групи користувачів

У Таблиці 4.7 наведено фактори можливостей ринкового впровадження проекту, серед яких: стрімкий ріст попиту на інструменти обробки медичних даних, поява нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг; було також запропоновано можливі реакції компанії.

У наступній таблиці наведено проведення аналізу сильних та слабких сторін стартап-проекту, факторами конкурентоспроможності виступили такі: наявність можливості впровадження як бібліотеки, наявність REST API, швидкість, гнучкість.

Таблиця 4.8 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з нашим підприємством*						
			-3	-2	-1	0	1	2	3
1.	Швидкість	20							+
3.	Гнучкість	15						+	

4.	Наявність можливості впровадження як бібліотеки	15					+		
5.	REST API	10					+		

4.3 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 9).

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 4.9 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1.	Дослідницькі групи, університети	Спрощення роботи з моделлю побудови embeddings	Великий	Існує 3 конкуренти, які надають схожі, але гірші рішення.	Швидкість, гнучкість
2.	Великі відео-сервіси	Спрощення роботи з побудовою рекомендацій	Великий		Можливість інтеграції в уже існуючі системи завдяки REST API, можливість використання як бібліотеки
Які цільові групи обрано: обираємо дослідницькі групи, університети та великі відео-сервіси					

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку. Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку. За М. Портером, існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового ринку та типом конкурентної переваги, що має бути реалізована на ринку (за витратами або визначними якостями товару).

Отже, проілюструвати базову стратегію розвитку можна у виляді Таблиці 4.10

Таблиця 4.10 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспромо жні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1.	Створення швидкої та гнучкої бібліотеки для отримання точкової інформації про відео	Ринкове позиціонування	Можливість інтеграції в уже існуючі системи завдяки REST API, бібліотеці, швидкість, гнучкість	Диференціація

Було обрано таку альтернативу розвитку проекту: створення веб-сервісу та бібліотеки використовуючи PyTorch, Python Flask, React/Redux, адже завдяки цим технологіям можна досягнути ключових конкурентноспроможних позицій кінцевого продукту.

Таблиця 4.11 - Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*
1.	Ні	Так	Побудова моделі побудови video embeddings	Зайняття конкурентної ніші

Отже, було визначено базову стратегію конкурентної поведінки як зайняття конкурентної ніші.

Визначимо стратегію позиціонування у Таблиці 4.12, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.12 - Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап- проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	Швидка, гнучка з бібліотека з можливістю	Диференціація	Швидка, гнучка з бібліотека з	Швидкість, гнучкість, API

	доступу через REST API		можливістю доступу через REST API	

4.4 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.13 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.13 - Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1.	Швидкість	Бібліотека на порядок швидша ніж інші	Перевага у швидкості.
2.	Гнучкість	Можливість використовувати бібліотеку для отримання будь-яких даних з мета-даних відео	Користувачі отримують гнучкий інструмент для отримання точкової інформації
3.	Наявність можливості впровадження як бібліотеки	Можливість використання звичним способом	Звичний спосіб використання, легкий перехід
4.	REST API	Використання в будь-яких архітектурах	Незалежність платформи

Далі у Таблиці 14 проілюстрована трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 4.14 - Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Веб-сервіс, що надає доступ до рекомендаційного сервісу за допомогою HTTP запитів, дозволяє працювати зі HTTP-ендпоїнтами та надає можливості логічного виведення та хмарного розгортання</td></tr>
<tr><td rowspan="5">ІІ. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>Швидкість
Гнучкість
Наявність можливості впровадження як бібліотеки
REST API</td><td>1.Нм
2.Нм
3.Нм
4.Нм</td><td>1.Технологічна
2.Технологічна
3.Технологічна
4.Технологічна</td></tr>
<tr><td colspan="3">Якість: згідно до стандарту ISO 4444 буде проведено тестування</td></tr>
<tr><td colspan="3">Маркування відсутнє</td></tr>
<tr><td colspan="3">Компанія: “VidRecommend”</td></tr>
<tr><td rowspan="2">ІІІ. Товар із підкріплюнням</td><td colspan="3">1-місячна пробна безкоштовна версія</td></tr>
<tr><td colspan="3">Постійна підтримка для користувачів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: патент</td></tr>
</table>

Було описано три рівні моделі товару, з чого можна зробити висновок, що основні властивості товару у реальному виконанні є нематеріальними та технологічними. Також було надано сутність та складові товару у задумці та товару з підкріпленням.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. У даному випадку найбільш вірогідним гарантом буде патент.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.15). Аналіз проводиться експертним методом.

Таблиця 4.15 - Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники, грн.*	*Рівень цін на товари-аналоги, грн.*	*Рівень доходів цільової групи споживачів, грн.*	*Верхня та нижня межі встановлення ціни на товар/послугу, грн.*
1.	45000	38000	150000	35000-40000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.16).

Таблиця 4.16 - Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Придбання підписки та оплата щомісячних внесків для продовження ліцензії	Продаж	0(напряму), 1(через одного посередника)	Власна та через посередників

Отже, система приноситиме прибуток завдяки щомісячним внескам для подовження ліцензії та придбанням підписок, продаж буде виконуватись напряму або через одного посередника.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.17).

Таблиця 4.17 - Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуютьс я цільові клієнти*	*Ключові позиції, обрані для позиціону вання*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1.	Придбання ліцензії на користування в мережі Інтернет, щомісячне її продовження, користування сервісом у хмарі або ж на власних серверах.	Інтернет	Інтеграція , хмарне розгортан ня, HTTP-ендпойнт	Показати переваги сервісу, у тому числі і перед конкурентами	Демо-ролик із використан ня, рекламні оголошення на популярних сайтах.

Отже, в Таблиці 4.17 наведено концепцію маркетингових комунікацій, було визначено, що придбання ліцензії на користування буде здійснюватись в мережі Інтернет, необхідним буде щомісячне її продовження, користування сервісом можливе у хмарі або ж на власних серверах.

4.5 Висновки

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Також, варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами. Для успішного виходу на ринок у продукту повинні бути наступні характеристики:

- наявність універсального API;
- можливість використання як бібіліотеки;

- швидкість;
- гнучкість.

В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

Було визначено такі сильні сторони: наявність універсального API , можливість використання як бібіліотеки, швидкість, гнучкість.

Можливості для виходу на ринок включають стрімкий ріст попиту на інструменти обробки медичних даних, можливість впровадження нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг. Наявні такі фактори загроз: конкуренція, зміна потреб користувачів, надходження на ринок альтернативних продуктів, уповільнення росту ринку.

ВИСНОВКИ

В результаті виконання роботи було побудовано рекомендаційну систему із використанням методів глибокого навчання, а саме алгоритму пошуку семантичних зв’язків між словами Word2Vec. Було побудовано модель SkipGram та нейронну мережу, яка будує векторні представлення відео у всьому корпусі відео-файлів для подальшого використання при побудові рекомендацій.

Також під час роботи над магістерською дисертацією було детально розглянуто основні засоби реалізації методів глибокого навчання, а саме – побудова нейронних мереж. Було проаналізовано структуру та методи побудови нейронних мереж та їх типів (згорткові, рекурентні мережі), їх переваги та недоліки.

Під час дослідження було детально розглянуто алгоритм пошуку семантичних зв’язків між словами Word2Vec з використанням алгоритмів SkipGram та CBoW. Під час роботи над даним алгоритмом було також проаналізовано алгоритми, що передували даному, а саме – алгоритми BoW, One-hot Encoding та TF-IDF. Саме з цих алгоритмів було виведено Word2Vec. Також були розглянуті математичні методи оптимізації звичайного Word2Vec: Negative Sampling та Hierarchical SoftMax.

Після огляду теоретичних джерел було зроблено огляд використання теорії на практиці, а саме – приклади реалізації рекомендаційних алгоритмів великих та відомих інтернет сервісів Airbnb та YouTube (Google). В результаті аналізу рекомендаційних систем цих сервісів було зроблено висновок, що чим менший сервіс, тим більший простір для побудови експериментів з використанням сучасних ідей в сфері машинного навчання.

Таким чином Airbnb має змогу використовувати усі напрацювання сучасних теоретиків NLP в сфері рекомендацій. Їх поточна система містить початкову реалізацію ідеї використання Word2Vec для побудови векторних представлень рекламних оголошень для кращого ранжування результатів пошуку. Звичайно, дана система ще тестується, але Airbnb оприлюднив пристойні результати, що означає успішне використання даної моделі для побудови рекомендацій.

YouTube навпаки через свій розмір (>1,000,000,000 відео та користувачів) не має змоги тестувати нові ідеї, оскільки навіть мала зміна сервісу вплине на інші. Хоча їх рекомендаційна система побудована на основі згорткових нейронних мереж з використанням ReLU та вони все таки використовують сесії користувачів для покращення результатів побудови ранжованого пошуку рекомендацій.

Під час розробки рекомендаційної системи було проаналізовано 2 основних засоби реалізації production-ready рішень для побудови експериментів з великими даними, а саме – PyTorch та TensorFlow. Після детального аналізу цих бібліотек було зроблено висновок, що PyTorch краще підходить для побудови невеликих експериментів, є більш зручним для роботи з даними та краще виконує такі типові задачі, як реалізація алгоритму градієнтного спуску або побудова згорткових нейронних мереж тощо.

Після аналізу засобів було детально проаналізовано алгоритм побудови моделі Word2Vec з інженерного боку для пошуку семантичних зв’язків між словами, оскільки він став ключовою ідеєю для побудови рекомендаційної системи. В результаті аналізу даної моделі було виявлено, що семантичні зв’язки між словами можна шукати за допомогою алгоритму SkipGram.

Також було побудовано рекомендаційну систему для великого корпусу відео на основі SkipGram моделі. Після побудови моделі було створено рекомендаційну систему, вбудовану у веб-додаток, створений за допомогою Flask (back-end API) та React/Redux (UI front-end) для наочної демонстрації результатів дослідження. З результатів роботи рекомендаційної системи було зроблено висновок, що система поводить себе коректно, рекомендації – релевантні. Тому можна використовувати поточне рішення для різного роду медіа-платформ для збільшення конверсії тощо.

Після побудови даної рекомендаційної системи було виявлено можливі шляхи оптимізації роботи алгоритму та покращення релевантності результатів. По-перше, можна використати більш оптимізовані методи Negative Sampling / Hierarchical SoftMax замість звичайного SkipGram. По-друге, можна окрім лише одних ID відео використовувати й інші мета-дані відео: назва, короткий опис, ключові слова, оцінки

тощо. Якщо піти ще далі, то можна звести дану NLP задачу до розв’язку комбінованої NLP + Computer Vision задачі, якщо до моделі додати контент відео в закодованому вигляді, але це може значно вплинути на якість розробки додатку, оскільки в такому випадку модель буде переускладнено.

ПЕРЕЛІК ПОСИЛАНЬ

1. Linden G. Item-to-Item Collaborative Filtering. / Linden G., Smith B., and York J. – Los Alamitos, CA, USA: IEEE Internet Computing, 2003. – С. 76-80.
2. Neural Networks Tutorial – A Pathway to Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: http://adventuresinmachinelearning.com/neural-networks-tutorial/
3. Нейронні мережі – шлях до глибинного навчання [Електронний ресурс] – Режим доступу до ресурсу: https://codeguida.com/post/739
4. Штучна нейронна мережа [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Штучна_нейронна_мережа.
5. Згорткова нейронна мережа [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Згорткова_нейронна_мережа.
6. Efficient estimation of word representations in vector space / T. Mikolov, K. Chen, G. Corrado, J. Dean. // CoRR. – 2013. – abs/1301.3781.
7. Чудесный мир Word Embeddings: какие они бывают и зачем нужны? [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/company/ods/blog/329410/
8. Hierarchical Probabilistic Neural Network Language Model / Morin, F., & Bengio, Y // Aistats. – 2005. – №5. – С. 120-144.
9. Listing Embeddings for Similar Listing Recommendations and Real-time Personalization in Search Ranking [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/airbnb-engineering/listing-embeddings-for-similar-listing-recommendations-and-real-time-personalization-in-search-601172f7603e
10. Deep Neural Networks for YouTube Recommendations / Paul Covington, Jay Adams, Emre Sargin // RecSys ’16 September 15-19. – 2016. – Boston, MA, USA. – С. 15-19
11. TensorFlow [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/.
12. PyTorch [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/.

13. PyTorch or TensorFlow? [Електронний ресурс] – Режим доступу до ресурсу: https://www.kdnuggets.com/2017/08/pytorch-tensorflow.html
14. PyTorch vs TensorFlow – spotting the difference [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b
15. Gradient-based learning applied to document recognition / Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. // Proceedings of the IEEE. – 1998. – №86(11). – С. 2278–2324.
16. Hu S. Video2Vec: Learning semantic spatio-temporal embeddings for video representation / S. Hu, Y. Li, B. Li. // International Conference on Pattern Recognition (ICPR). – 2016. – №23. – С. 811–816.
17. Word2Vec Tutorial - The Skip-Gram Model [Електронний ресурс] – Режим доступу до ресурсу: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
18. Word2Vec Tutorial Part 2 - Negative Sampling [Електронний ресурс] – Режим доступу до ресурсу: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/
19. Welcome to Flask [Електронний ресурс] – Режим доступу до ресурсу: http://flask.pocoo.org/docs/1.0/
20. Redux Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

SkipGram Model Train

```
import pickle

import gc



import torch

from torch.autograd import Variable

import torch.nn as nn

import torch.nn.functional as F



class SkipGramModel(nn.Module):

    """Skip gram model of gifid2vec.

    Attributes:

        emb_size: Embedding size.

        emb_dimention: Embedding dimention, usually 32.

        u_embedding: Embedding for center gif.

        v_embedding: Embedding for neibor gifs in session.

    """



    def __init__(self, emb_size, emb_dimension):

        """Initialize model parameters.

        Apply for two embedding layers.

        Initialize layer weight

        Args:

            emb_size: Embedding size.

            emb_dimention: Embedding dimention, typically 32.

        Returns:

            None

        """

        super(SkipGramModel, self).__init__()
```

```
            self.emb_size = emb_size
            self.emb_dimension = emb_dimension
            self.u_embeddings = nn.Embedding(emb_size, emb_dimension, sparse=True)
            self.v_embeddings = nn.Embedding(emb_size, emb_dimension, sparse=True)
            self.init_emb()


        def init_emb(self):
            """Initialize embedding weight like word2vec.
            The u_embedding is a uniform distribution in [-0.5/em_size, 0.5/emb_size], and the elements of v_embedding are zeroes.
            Returns:
                None
            """
            initrange = 0.5 / self.emb_dimension
            self.u_embeddings.weight.data.uniform_(-initrange, initrange)
            self.v_embeddings.weight.data.uniform_(-0, 0)


        def forward(self, pos_u, pos_v, neg_v):
            """Forward process.
            As pytorch designed, all variables must be batch format, so all input of this method is a list of gif id.
            Args:
                pos_u: list of center gif id indexes for positive gif pairs.
                pos_v: list of neibor gif id indexes positive gif pairs.
                neg_u: list of center gif id indexes negative gif pairs.
                neg_v: list of neibor gif id indexes negative gif pairs.
            Returns:
                Loss of this process, a pytorch variable.
            """
            emb_u = self.u_embeddings(pos_u)
            emb_v = self.v_embeddings(pos_v)
            score = torch.mul(emb_u, emb_v).squeeze()
            score = torch.sum(score, dim=1)
            score = F.logsigmoid(score)
            neg_emb_v = self.v_embeddings(neg_v)
```

```
        neg_score = torch.bmm(neg_emb_v, emb_u.unsqueeze(2)).squeeze()
        neg_score = F.logsigmoid(-1 * neg_score)
        return -1 * (torch.sum(score) + torch.sum(neg_score))
```

Лістинг тренування моделі SkipGramTraining

```
import pickle
import sys
import os

import numpy
import torch
import nmslib
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from tqdm import tqdm

from model_train.dataset.skip_gram_dataset_reader import SkipGramDatasetReader
from model_train.models.skip_gram_model import SkipGramModel
from model_train.utils import ensure_dir


class SkipGramTrainer:
    def __init__(self,
                 input_file_name,
                 output_folder,
                 emb_dimension=32,
                 batch_size=512,
                 window_size=3,
                 iteration=5,
                 initial_lr=0.01,
                 min_count=5):
        """Initilize class parameters.
        Args:
```

```
                input_file_name: Name of a text data from file. Each line is a gif ids sequence
splited with space.

                output_folder: Name of the final embedding folder.

                emb_dimention: Embedding dimention, typically 32.

                batch_size: The count of gif id pairs for one forward.

                window_size: Max skip length between gifs.

                iteration: Control the multiple training iterations.

                initial_lr: Initial learning rate.

                min_count: The minimal gif frequency, gifs with lower frequency will be
filtered.

            Returns:

                None.

            """

            self._data = SkipGramDatasetReader(input_file_name, min_count)

            self._output_folder = output_folder

            self._emb_size = len(self._data.gif2id)

            self._emb_dimension = emb_dimension

            self._batch_size = batch_size

            self._window_size = window_size

            self._iteration = iteration

            self._initial_lr = initial_lr

            self._skip_gram_model = SkipGramModel(self._emb_size, self._emb_dimension)

            self._use_cuda = torch.cuda.is_available()

            if self._use_cuda:

                self._skip_gram_model.cuda()

            self._optimizer          =          optim.SGD(self._skip_gram_model.parameters(),
lr=self._initial_lr)


        def train(self):

            """Multiple training.

            Returns:

                None.

            """

            pair_count = self._data.evaluate_pair_count(self._window_size)

            batch_count = self._iteration * pair_count / self._batch_size

            process_bar = tqdm(range(int(batch_count)), smoothing=1.0)
```

```
            for i in process_bar:
                pos_pairs = self._data.get_batch_pairs(self._batch_size, self._window_size)
                neg_v = self._data.get_neg_v_neg_sampling(pos_pairs, 5)
                pos_u = [pair[0] for pair in pos_pairs]
                pos_v = [pair[1] for pair in pos_pairs]

                pos_u = Variable(torch.LongTensor(pos_u))
                pos_v = Variable(torch.LongTensor(pos_v))
                neg_v = Variable(torch.LongTensor(neg_v))
                if self._use_cuda:
                    pos_u = pos_u.cuda()
                    pos_v = pos_v.cuda()
                    neg_v = neg_v.cuda()

                self._optimizer.zero_grad()
                loss = self._skip_gram_model.forward(pos_u, pos_v, neg_v)
                loss.backward()
                self._optimizer.step()

                process_bar.set_description(f'Loss:         {loss.data[0]:.4f},         lr:
{self._optimizer.param_groups[0]["lr"]:.6f}')
                if i * self._batch_size % 100000 == 0:
                    lr = self._initial_lr * (1.0 - 1.0 * i / batch_count)
                    for param_group in self._optimizer.param_groups:
                        param_group['lr'] = lr
            self._save_embedding()

        def _save_embedding(self):
            ensure_dir(self._output_folder)

            if self._use_cuda:
                embedding = self._skip_gram_model.u_embeddings.weight.cpu().data.numpy()
            else:
                embedding = self._skip_gram_model.u_embeddings.weight.data.numpy()
```

```
        gif_to_vec = {}
        for gif, index in self._data.gif2id.items():
            gif_to_vec[gif] = embedding[index]

        with open(os.path.join(self._output_folder, 'vocabs'), 'wb') as output_file:
            pickle.dump(
                {
                    'gif_to_vec': gif_to_vec,
                    'index_to_gif': self._data.gif2id
                },
                output_file
            )

        del gif_to_vec

        nms_index = nmslib.init(method='hnsw', space='cosinesimil')
        nms_index.addDataPointBatch(embedding)
        nms_index.createIndex()

        nmslib.saveIndex(nms_index, os.path.join(self._output_folder, 'nms_index'))


if __name__ == '__main__':
    skipgram = SkipGramTrainer(
        input_file_name=os.getenv('DATASET_FILE'),
        output_folder=os.getenv('OUTPUT_PATH')
    )
    skipgram.train()
```

Лістинг побудови API рекомендаційної системи

```
import requests
import os

class Gif(object):
```

```
        def __init__(self, gif_id, score):
            self.id = gif_id
            self.score = score

            self.__load()

        def serialize(self):
            return self.__dict__

        def __load(self):
            response_json = requests.get(self.__metadata_url()).json()['data']

            self.gif_url = response_json['images']['source']
            self.giphy_url = response_json['url']
            self.fixed_height_url = response_json['images']['fixed_height']
            self.tags = response_json['tags']
            self.featured_tags = response_json['featured_tags']
            self.import_datetime = response_json['import_datetime']
            self.title = response_json['title']
            self.web_entities = []
            self.google_text_result = None
            self.__load_extended(response_json.get('extended', []))

        def __metadata_url(self):
            return f'{os.getenv("METADATA_SERVICE_URL", "http://metadata-
qa.giphy.tech")}/metadata/{self.id}?extended=true'

        def __load_extended(self, extended):
            if len(extended) == 0:
                return

            if 'google_web_result' in extended[0]:
                self.web_entities = [we['description'] for we in
extended[0]['google_web_result'].get('web_entities', [])]
```

```
        if 'google_text_result' in extended[0]:
            self.google_text_result = extended[0]['google_text_result']
```

Лістинг побудови UI рекомендаційної системи

```
import { createStore, applyMiddleware, compose } from 'redux'
import { routerMiddleware } from 'react-router-redux'
import thunk from 'redux-thunk'
import createHistory from 'history/createBrowserHistory'
import rootReducer from './modules'

export const history = createHistory()

const initialState = {}
const enhancers = []
const middleware = [
  thunk,
  routerMiddleware(history)
]

if (process.env.NODE_ENV === 'development') {
  const devToolsExtension = window.devToolsExtension

  if (typeof devToolsExtension === 'function') {
    enhancers.push(devToolsExtension())
  }
}

const composedEnhancers = compose(
  applyMiddleware(...middleware),
  ...enhancers
)

const store = createStore(
  rootReducer,
  initialState,
```

```
  composedEnhancers
)

export default store

import React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
import { ConnectedRouter } from 'react-router-redux';
import store, { history } from './store';
import App from './containers/app';

import './index.css';

const target = document.querySelector('#root');

render(
  <Provider store={store}>
    <ConnectedRouter history={history}>
      <div>
        <App store={store} />
      </div>
    </ConnectedRouter>
  </Provider>,
  target
);

// In production, we register a service worker to serve assets from local cache.

// This lets the app load faster on subsequent visits in production, and gives
// it offline capabilities. However, it also means that developers (and users)
// will only see deployed updates on the "N+1" visit to a page, since previously
// cached resources are updated in the background.

// To learn more about the benefits of this model, read https://goo.gl/KwvDNy.
```

```
// This link also includes instructions on opting out of this behavior.

const isLocalhost = Boolean(
  window.location.hostname === 'localhost' ||
    // [::1] is the IPv6 localhost address.
    window.location.hostname === '[::1]' ||
    // 127.0.0.1/8 is considered localhost for IPv4.
    window.location.hostname.match(
      /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/
    )
);

export default function register() {
  if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator) {
    // The URL constructor is available in all browsers that support SW.
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location);
    if (publicUrl.origin !== window.location.origin) {
      // Our service worker won't work if PUBLIC_URL is on a different origin
      // from what our page is served on. This might happen if a CDN is used to
      // serve assets; see https://github.com/facebookincubator/create-react-
app/issues/2374
      return;
    }

    window.addEventListener('load', () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        // This is running on localhost. Lets check if a service worker still exists or
not.
        checkValidServiceWorker(swUrl);

        // Add some additional logging to localhost, pointing developers to the
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
```

```
          console.log(
            'This web app is being served cache-first by a service ' +
              'worker. To learn more, visit https://goo.gl/SC7cgQ'
          );
        });
      } else {
        // Is not local host. Just register service worker
        registerValidSW(swUrl);
      }
    });
  }
}

function registerValidSW(swUrl) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        installingWorker.onstatechange = () => {
          if (installingWorker.state === 'installed') {
            if (navigator.serviceWorker.controller) {
              // At this point, the old content will have been purged and
              // the fresh content will have been added to the cache.
              // It's the perfect time to display a "New content is
              // available; please refresh." message in your web app.
              console.log('New content is available; please refresh.');
            } else {
              // At this point, everything has been precached.
              // It's the perfect time to display a
              // "Content is cached for offline use." message.
              console.log('Content is cached for offline use.');
            }
          }
        };
```

```
      };
    })
    .catch(error => {
      console.error('Error during service worker registration:', error);
    });
}

function checkValidServiceWorker(swUrl) {
  // Check if the service worker can be found. If it can't reload the page.
  fetch(swUrl)
    .then(response => {
      // Ensure service worker exists, and that we really are getting a JS file.
      if (
        response.status === 404 ||
        response.headers.get('content-type').indexOf('javascript') === -1
      ) {
        // No service worker found. Probably a different app. Reload the page.
        navigator.serviceWorker.ready.then(registration => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        // Service worker found. Proceed as normal.
        registerValidSW(swUrl);
      }
    })
    .catch(() => {
      console.log(
        'No internet connection found. App is running in offline mode.'
      );
    });
}

export function unregister() {
```

```
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.ready.then(registration => {
      registration.unregister();
    });
  }
}
```